

Technologies Web élémentaires : HTTP, HTML, CGI, Javascript et Ajax

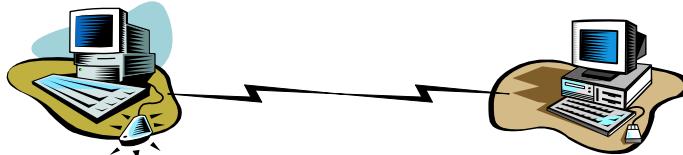
Claude VILVENS

claude.vilvens@hepl.be

<http://haute-ecole.provincedeliege.be/>

Sommaire

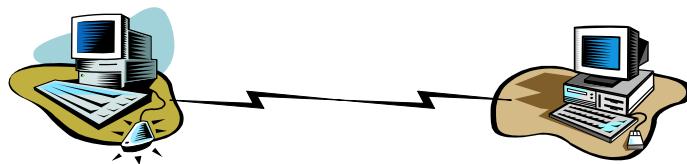
Introduction



HTTP : le protocole de base

I. Les principaux aspects du protocole HTTP

1. Le protocole client-serveur du WEB	
1.1 Le serveur	2
1.2 Le client	3
2. Les caractéristiques de base du protocole	
2.1 Les objectifs	5
2.2 L'évolution	5
2.3 La notion de ressource	5
3. La codification MIME	6
4. Un protocole requête/réponse	7
5. Une visualisation du trafic réseau	
5.1 Une communication en 10 paquets TCP	9
5.2 La requête HTTP	10
5.3 La réponse HTTP	10
6. La forme générale des messages du protocole HTTP	11
7. La requête	
7.1 La forme et les méthodes	12
7.2 Quelques champs du header de requête	13
7.3 La liste des headers de requête	14
8. La réponse	
8.1 La forme et les méthodes	15
8.2 Quelques champs du header de réponse	16
8.3 La liste des headers de réponse	17
9. Les entités	
9.1 Une réponse qui encapsule une information	17
9.2 La liste des headers d'entité	18
10. La sécurité http	
10.1 L'authentification basique	18
10.2 L'authentification par digest	18
10.3 HTTPS	19
11. Les cookies	19



HTML : les pages statiques

II. Le langage HTML : les bases

1. Le langage HTML	22
2. Les balises, les marqueurs	22
3. Structure de base d'un document HTML	23
4. Création d'une page HTML	24
5. Encoder du texte	27
6. Placer des titres	28
7. D'autres possibilités de mises en forme	31
8. Les propriétés	31
9. Les tags META	33
10. Au-delà de HTML	
10.1 SGML	36
10.2 XML	36
10.3 XHTML	37

III. Les liens

1. L'hypertexte de base	38
2. La balise de liens	41
3. Les chemins et les URLs	42
4. Les signets	43

IV. Les images

1. Les images utilisées sur le WEB	47
2. L'insertion d'une image	47
3. La taille mémoire des images	
3.1 Position du problème	50
3.2 La taille	51
3.3 Le format	52
3.4 Les couleurs	52
4. Le texte accompagnant les images	52
5. Les images déclencheurs de lien	54
6. Les images référençantes	56
7. Trouver ou générer des images	58

V. Les listes et les tableaux

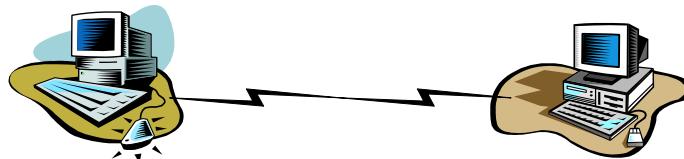
1. Les listes à puces	59
2. Les listes imbriquées	60
3. Les listes numérotées	62
4. Les glossaires	63
5. Les menus	64
6. Les tableaux de base	65
7. Quelques compléments aux tableaux de base	
7.1 La mise en page du tableau	68
7.2 Les titres de colonnes	69
7.3 Le titre général du tableau	71
7.4 Modifier le tableau	72
8. Les tableaux complexes	72

VI. Les frames

1. L'objectif	75
2. La définition d'une page à frames	76
3. Le document cible	78
4. Le document des références	79

VII. Les feuilles de style CSS

1. Une plus grande souplesse	82
2. Les règles de style en interne	82
3. Les règles de style dans un fichier externe	85
4. Les classes de style	86
5. Les styles et les balises div	
5.1 Une mise en page à 3 zones	89
5.2 Une mise en page à 4 zones	91
6. Les styles de tableaux	93



HTTP & HTML : les pages dynamiques et interactives

VIII. Les formulaires et les CGIs

1. L'objectif	97
2. Les éléments de base d'un formulaire	98
3. Les boîtes de listes	100
4. La mise en place du formulaire	101
5. Les gateways	103
6. Un exemple de gateway	104
7. Le CGI d'un formulaire	107

IX. Une petite introduction à JavaScript

1. Un langage de script	112
2. Les "objets" implicites de JavaScript	113
3. L'objet window	114
4. L'objet document et son contenu	114
5. Un script basique	115
6. Les mots réservés de JavaScript	118
7. Les scripts externes	119
8. Le traitement des événements	121
9. Un exemple de validation de formulaire	123
10. Une introduction à AJAX	
10.1 Présentation des concepts Web 2.0	127
10.2 Un modeste exemple Ajax synchrone	133
10.3 Un modeste exemple Ajax asynchrone texte	135
10.4 Un modeste exemple Ajax asynchrone xml	137

X. Une petite introduction à JavaScript

1. Encore une nouvelle norme ?	139
2. Principales nouveautés apportées par HTML 5	139
3. Le remplacement des frames : les balises sémantiques de navigation	141
4. Les formulaires avec validation de champs	145

Ouvrages consultés

Introduction



Les applications du protocole **HTTP** (HyperText Transfer Protocol) sont sans doute celles qui sont le plus connues dans le grand public parmi toutes celles qui utilisent les protocoles de la suite TCP/IP. En particulier, les célèbres pages WEB, avec le langage **HTML** (HyperText Markup Language) qui les sous-tend, est familier à tous les internautes en mal de site perso.

Les notes présentées ici correspondent donc au besoin de clarifier les idées et d'énoncer des faits qui, parfois, sont bien connus et parfois, au contraire, semblaient clairs mais en fait ne le sont pas. On trouvera donc ici, côté à côté, des portes ouvertes enfoncées et des caches secrètes dévoilées ...

Nous nous attaquerons tout d'abord au protocole **HTTP** proprement dit. Il n'est pas très difficile et sa structure se comprend assez facilement – plus facilement d'ailleurs que les RFCs qui le concernent : leur aridité est sans rapport avec les belles pages WEB que le protocole contribue à amener jusqu'à nous !

La partie médiane de l'ouvrage concerne le langage **HTML**. Nous l'étudierons en tant que tel, dans sa version standard, avec un simple (et antique ;-) !) browser comme Netscape (en sa version 7.0) et un simple (et préhistorique ;-) !) éditeur HTML comme Netscape Composer (version 7.*). L'utilisation de produits d'édition de pages WEB plus évolués, particulièrement **Macromedia Dreamweaver MX**, est évidemment conseillée (et pour tout dire, indispensable) d'un point de vue professionnel même si certains de ces logiciels, comme MS FrontPage, présentent des caractéristiques propriétaires, certaines de leurs balises étant exotiques.

La dernière partie est sans doute la plus intéressante pour les programmeurs. Ils y trouveront d'abord le traitement de requêtes clients par le serveur au moyen des **CGIs** (Common Gateway Interface – connaissance du langage C requise). On peut réaliser ce traitement au moyen des servlets, mais la connaissance du langage Java est nécessaire et ne seront pas abordées ici¹. Côté client, le dernier chapitre nous offrira une petite incursion dans le monde **JavaScript**, avec une timide incursion dans la philosophie Ajax.

Il est supposé que le lecteur de ces pages possède les connaissances de base de TCP/IP. Moyennant cette petite condition, tout ceci est en définitive fort simple ... ;-)

Claude Vilvens

¹ voir "Langage Java (III) : Programmation des applications WEB" – du même auteur ...

I. Les principaux aspects du protocole HTTP



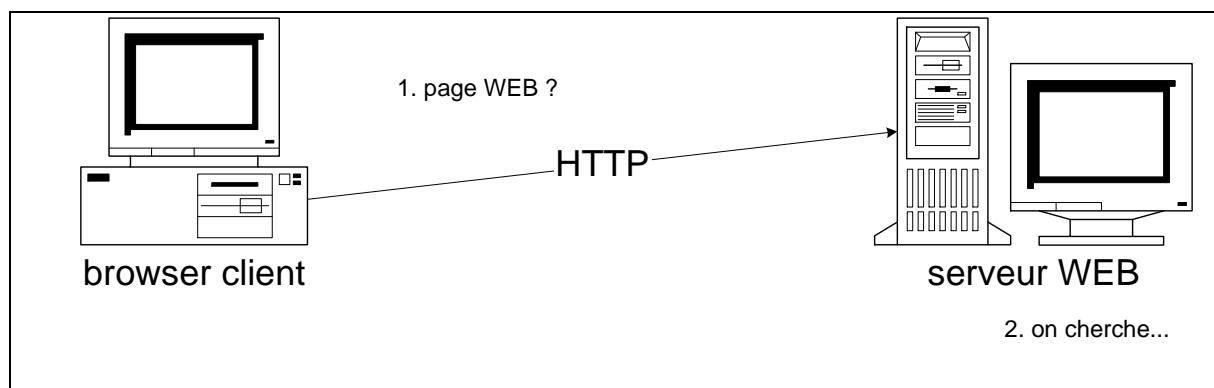
Il est bon de suivre sa pente pourvu que ce soit en montant.

(A.Gide, Les Faux-Monnayeurs)

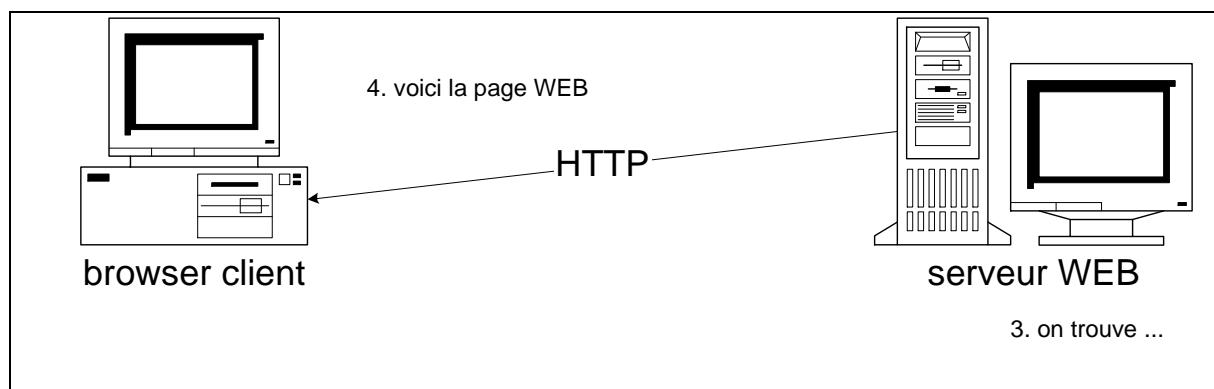
1. Le protocole client-serveur du WEB

1.1 Le serveur

Un serveur HTTP, mieux connu sous le nom de "serveur WEB", est un programme qui se trouve sur un ordinateur d'Internet. Par extension, le terme "serveur WEB" désigne aussi la machine hôte de ce programme. Celui-ci attend qu'un autre ordinateur client, probablement muni d'interface de navigation désigné sous le nom de "browser", vienne s'y connecter et lui présente une **requête**, qui consiste en général en la demande d'une page sous forme de fichier :



Une fois cette requête reçue, le serveur tente de localiser le fichier et, s'il le trouve, il procède à son envoi - dans le cas contraire, il envoie un message d'erreur :



On se trouve manifestement dans le contexte du "modèle client-serveur". Le browser est donc alors le "client WEB".

Toute communication entre ordinateurs ne peut se faire qu'en utilisant un *protocole*, c'est-à-dire un ensemble de règles à observer pour assurer une communication valable. Dans le cas des pages WEB, le client et le serveur communiquent au moyen du protocole HTTP

(**Hypertext Transfer Protocol**), spécialement conçu pour le transfert de documents hypertextes sur le WEB. Un serveur WEB est encore appelé un **HTTPD**, du nom du **HTTP Deamon**, programme "démon" travaillant en tâche de fond. Un démon attend une requête; il se réveille quand il la reçoit, la traite, puis se rendort.

Les logiciels HTTPD les plus utilisés sont historiquement ceux du CERN et de NCSA (National Center for Supercomputing Applications de l'Université de l'Illinois), mais surtout **Apache**.



Le serveur Web Apache est sans doute le plus célèbre et le plus utilisé des serveurs Web. Il existe en fait pour différentes plates-formes (HPUnix, Digital Unix, Linux, Windows, Mac OsX). Il représente plus ou moins 2/3 du marché des serveurs Web, notamment parce que c'est un projet libre. Dans sa version actuelle 2.0, il est multithread (il était auparavant multiprocessus)². Dans le monde des développeurs, c'est le serveur HTTP de IIS (Internet Information Services de Windows) et surtout Tomcat³, moteur à servlets mais aussi serveur Web, qui est très utilisé.

En fait, *le serveur accompagne le fichier transmis d'informations concernant ce fichier* (par exemple, s'il s'agit d'un fichier GIF, d'une séquence vidéo, etc). Le browser peut ainsi déterminer si il est capable de traiter lui-même le fichier ou s'il doit recourir à un assistant. Ce sont ces informations qui justifient la définition d'un protocole orienté vers la nature des données transmises ...

1.2 Le client

Dans le cadre qui nous intéresse ici, soit l'utilisation de pages hypertextes d'Internet, il est nécessaire pour l'utilisateur de disposer d'un programme spécial appelé **l'interface de navigation** ou **browser**⁴. Le rôle du browser est d'accéder aux informations se trouvant sur les serveurs et de l'interpréter pour la rendre présentable et donc effectivement accessible. En anticipant quelque peu sur le chapitre II, une page Web (ici, home-ferme-basique.html) a classiquement un aspect du type suivant :

home-ferme-basique.html

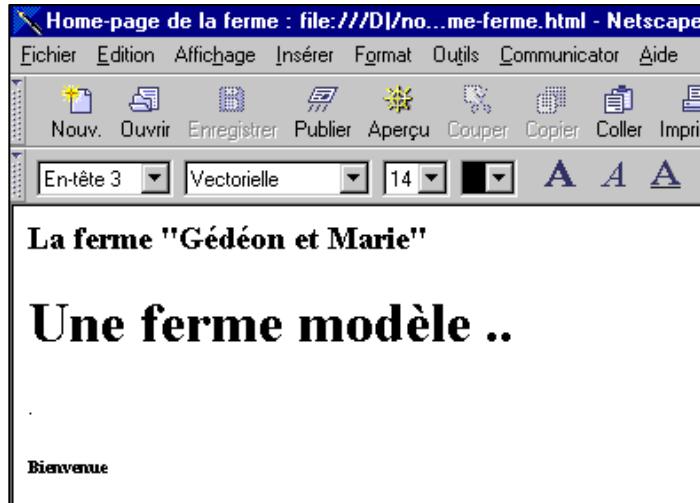
```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Home-page de la ferme</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
</head>
<body>
<!-- Version 1.0 - Rédigé par super-génie --&gt;
&lt;h3&gt;La ferme "G&amp;acute;d&amp;acute;on et Marie"&lt;/h3&gt;
&lt;h1&gt;Une ferme mod&amp;egrave;le ..&lt;/h1&gt;
&lt;h6&gt;Bienvenue&lt;/h6&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>
```

² pour plus de détails techniques, voir http://httpd.apache.org/docs/2.0/new_features_2_0.html

³ <http://jakarta.apache.org/tomcat>

⁴ les Français disent "navigateur" ou encore "butineur" (comme les abeilles ;-)) ...

où chaque symbole "<..>", comme par exemple <head> ou <h3>, est appelé un "**tag**" (ou "**balise**" en français). Elle se trouve sur le serveur qui fournira la page à la demande. L'utilisateur-client qui a demandé cette page a donc intérêt à interagir avec le serveur au moyen d'un browser qui est capable d'interpréter les tags HTML qui déterminent la manière d'afficher le contenu de la page, ce qui donne :



En l'absence de browser sur le client, c'est le fichier texte tel quel qui lui sera présenté ☺ ... En parallèle, le browser peut aussi comprendre d'autres protocoles, comme ftp.

Il existe un certain nombre de browsers, mais les deux plus performants sont Firefox (de Mozilla) et Internet Explorer (de Microsoft). Mais il en existe d'autres, comme Netscape (de Netscape Communications et, depuis peu, propriété d'AOL) :

<i>browser</i>	<i>environnement</i>
Mozilla-Firefox	UNIX, Windows, Mac, AIX
Internet Explorer	Windows
Opera	Windows, WAP, mobiles
Netscape	Windows, UNIX, Mac
Mosaic	Windows, Mac, UNIX

Netscape et Internet Explorer se sont disputé âprement la prédominance en termes de nombre d'utilisateurs, à l'avantage actuel du browser de Microsoft qui doit à présent tenir compte de Firefox. En fait, ils sont techniquement assez équivalents.

Comme on sait, Internet Explorer fait partie de la "suite" de logiciels de MS-Office, avec notamment, pour ce qui concerne Internet, le logiciel de messagerie MS-Outlook. De même, Netscape comporte :

- ◆ un browser proprement dit;
- ◆ un éditeur de pages HTML nommé "**composer**";
- ◆ un logiciel de messagerie électronique.

Dans la suite de ces notes, nous utiliserons le plus souvent Firefox ou Netscape, rien que pour ne pas donner l'impression d'avoir vendu notre âme au diable ;-)

2. Les caractéristiques de base du protocole

2.1 Les objectifs

Le protocole **HTTP** (HyperText Transfer Protocol) est un protocole de la couche application destiné à être utilisé au sein de systèmes d'information distribués, interactifs et multi-média.

Rien n'interdit de l'utiliser dans d'autres domaines, mais il est clairement orienté vers le typage et la négociation des représentations de données entre deux machines communiquant par un réseau TCP/IP. Le fait de s'adapter aux représentations utilisables sur les machines permet la construction de systèmes indépendamment du type des données qui seront transférées entre les composantes de ces systèmes.

On peut remarquer que si TCP est le protocole de transport supposé (avec le port **80** comme port par défaut), le choix de ce protocole n'est cependant pas une obligation. En fait, HTTP suppose simplement se reposer sur un protocole de transport fiable.

2.2 L'évolution

La première version à être effectivement utilisée en 1990 était HTTP/0.9. En fait, il s'agissait d'un simple protocole de transfert de données brutes.

HTTP/1.0 (RFC 1945) est beaucoup plus évolué, puisqu'il utilise des messages utilisant le format **MIME** (format des messages utilisés par la messagerie électronique, donc par le protocole SMTP). Une caractéristique essentielle de tels messages est qu'ils contiennent des informations sur la nature des données transmises (on parle encore de "meta-information") ainsi que des directives permettant de paramétrier la requête ou la réponse (on parle encore de "modificateurs sémantiques"). Cette version est définie dans la RFC 1945.

Avec le développement des techniques du WEB (proxies, cache, etc), il est apparu qu'il fallait encore améliorer le protocole pour que chaque machine intervenante puisse déterminer les possibilités, de ce point de vue, des autres machines. HTTP/1.1 (RFC 2068) a cette prétention.

2.3 La notion de ressource

Le terme "ressource" est en fait un terme assez vague désignant aussi bien des données (dans des formats divers) que des services. La seule exigence est que l'objet du désir puisse être désigné par un **URI**, c'est-à-dire un **Uniform Resource Identifier**. La forme la plus classique est l'**URL** (**Uniform Resource Locator**) bien connue des surfeurs du WEB. Ainsi :

www.genie-vil.org/qualites/beau.html

peut bien désigner un ensemble de données (soit une page HTML) tandis que

www.bucheron.be:8085

permet peut-être d'accéder aux fonctions d'administration d'un serveur – il s'agit alors ici d'un service, identifié plus spécifiquement par le numéro de port complémentaire (ici, 8085).

Quoi qu'il en soit, les URI peuvent exister sous une forme *relative* ou *absolue*, selon qu'ils font référence à un élément de base ou pas. Un URI absolu débutera toujours par un nom suivi d'un point.

3. La codification MIME

On entend par **MIME** (Multipurpose Internet Mail Extensions) les spécifications d'un protocole décrivant comment désigner la nature de données les plus diverses de manière à ce qu'elles puissent être envoyées par un système de messagerie électronique basé uniquement sur l'envoi de messages textes (ce qui était le cas aux débuts des e-mails). Ces spécifications sont détaillées dans les RFCs 2045-6-7-8-9.

L'objectif est donc de décrire les données contenues dans le corps du message de manière à ce que le **user-agent** (c'est-à-dire le logiciel derrière lequel se cache l'utilisateur) qui reçoit le message puisse mettre en place les dispositifs nécessaires pour lire ces données ou, à tout le moins, mettre en place un mécanisme quelconque de conversion. L'idée est donc qu'un message e-mail comporte, outre le message proprement dit, un header comportant divers champs décrivant au mieux le type de son contenu. Ainsi, un e-mail des plus simples aura pour en-tête :

From: James Vil < civilvens@prov-liege.be >	champs du protocole de messagerie
To: Francesca < francesca@skynet.be >	
Subject: Libre ce soir ?	
MIME-Version: 1.0	champs MIME
Content-type: text/plain	

Les deux principaux champs du header MIME sont donc :

- ◆ **le champ de version** (MIME-Version) - par exemple : MIME-Version: 1.0
- ◆ **le champ du type du contenu du message** (Content-Type) : il spécifie la nature des données contenues dans le message en fournissant, de manière normalisée, le type et le sous-type, avec éventuellement des informations complémentaires pour les types qui en réclament. Il existe 7 types de base :

type	définition	quelques sous-types courants
text	information textuelle	<ul style="list-style-type: none"> ◆ plain : texte non formaté – aucun software complémentaire n'est nécessaire pour appréhender l'entièreté du message – il est possible de spécifier le jeu de caractères par une clause charset : Content-type: text/plain; charset=us-ascii Content-type: text/plain; charset=iso-885-5 <ul style="list-style-type: none"> ◆ html : le texte comporte des tags HTML
image	bien sûr, il s'agit d'images au format gif ou jp(e)g	<ul style="list-style-type: none"> ◆ gif ◆ jpeg
audio	informations nécessitant un dispositif de son	<ul style="list-style-type: none"> ◆ basic
video	à votre avis ?	<ul style="list-style-type: none"> ◆ mpeg

application	informations binaires, à priori destinées à être exécutées par une application basée sur la messagerie	<ul style="list-style-type: none"> ◆ octet-stream : données binaires brutes – c'est le sous-type utilisé lorsqu'une application client ne connaît pas le type/sous-type précisé dans le message reçu ◆ postscript : pour transporter des informations codées en postscript
message	message faisant partie d'un autre message et susceptible de contenir des informations d'une autre nature que celle du message principal	<ul style="list-style-type: none"> ◆ rfc822 : selon la RFC de base de la messagerie électronique
multipart	données composées de groupes d'informations de types différents	<ul style="list-style-type: none"> ◆ mixed : les différentes composantes sont indépendantes et doivent être fournies dans un ordre précis ◆ alternative : les différentes composantes sont des versions différentes de la même information – le choix se fera selon les préférences déclarées du récepteur ◆ parallel : l'ordre des différentes parties n'a pas d'importance

Le champ Content-Type peut encore utiliser des types propriétaires, référencés par "X-...", encore qu'il soit plutôt recommandé de définir des sous-types des 7 types existants (référencés par "x-..."). Ainsi, par exemple, le mécanisme des servlets Java utilise de nouveaux sous-types du type application comme x-www-form-urlencoded ou x-java-serialized-object.

HTTP n'est pas, à proprement parler, "MIME-compliant". Mais il en fait cependant grand usage.

4. Un protocole requête/réponse

Comme déjà dit, le fonctionnement de base de HTPP est des plus simples :

un client crée une requête et l'envoie, le serveur répond et la transaction est terminée.

Le protocole est **sans état** [*stateless*], à la différence, par exemple, de tcp avec sa célèbre machine à nombre d'états fini : une fois la transaction terminée, il ne reste aucune trace de l'identification de l'utilisateur. Un serveur HTTP ne peut donc pas savoir qu'une série de requêtes proviennent toutes du même client.

Un **client** prend donc l'initiative de la communication en envoyant une requête au serveur. Cette requête comporte :

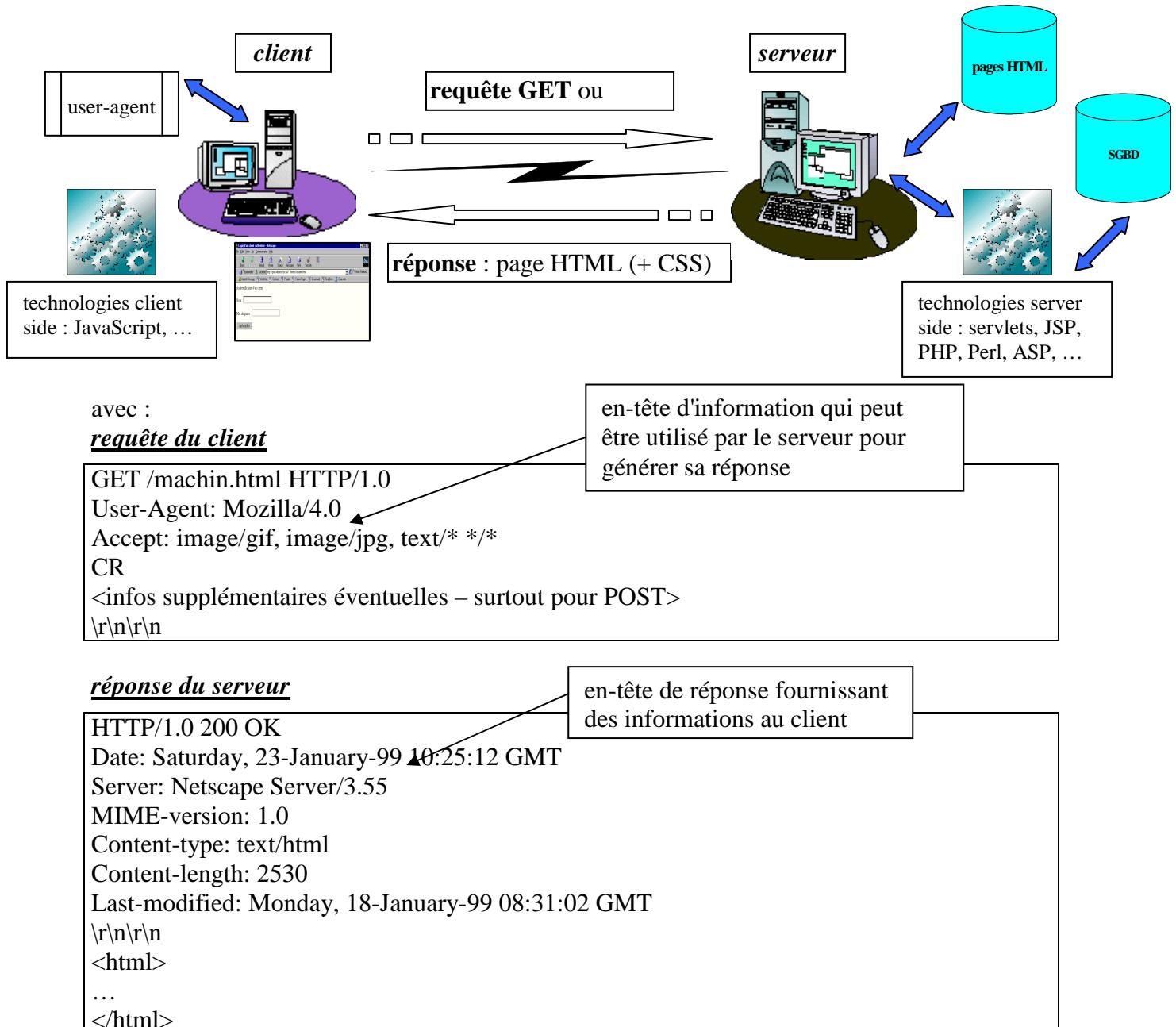
- ◆ la méthode utilisée pour cette requête (typiquement il s'agit des méthodes GET ou POST);
- ◆ l'URI de la ressource demandée;
- ◆ la version du protocole utilisée;
- ◆ un message MIME contenant diverses informations comme le type de données acceptés, des modificateurs de requête, etc.

On désigne encore sous le nom de "**user agent**" un tel client, qui peut être un **browser**, un éditeur, un **spider** (c'est-à-dire un robot logiciel passant en revue les sites WEB).

Le **serveur** répondra avec une "ligne d'état", c'est-à-dire un message précisant

- ◆ la version retenue du protocole;
- ◆ un code de succès ou d'erreur;
- ◆ un message MIME donnant des informations sur le serveur (comme les formats de données supportés).

Le dialogue peut ensuite s'engager plus en avant. Schématiquement, un exemple de requête serait :



On remarquera que la partie relevant du protocole (donc différente des données proprement dites) se termine toujours par une ligne vide. Evidemment, les choses peuvent être un peu plus complexes si l'on voit intervenir :

- ◆ **un proxy** : il s'agit d'un programme intermédiaire qui reçoit certaines requêtes initialement adressées au serveur WEB pour les traiter; en ce sens, il agit à la fois comme un client (il reçoit une requête) et comme un serveur (il renvoie une réponse)⁵; souvent, il travaille en conjonction avec un firewall, permettant les accès à Internet depuis l'arrière de celui-ci.
- ◆ **un gateway** (passerelle – vocable polymorphe s'il en est) : il s'agit d'un serveur qui sert d'intermédiaire pour un autre serveur; il considère la requête reçue comme si elle lui était destinée directement (à la différence d'un proxy); le client peut donc très bien ne pas se rendre compte qu'il converse avec un tel gateway.

En réalité, la réponse fournie l'est sous une représentation qui semble la meilleure. Malheureusement, cette "meilleure représentation" n'est pas la même pour tous les clients. En fait, elle doit être sélectionnée parmi toutes les formes possibles : on parle encore de "**Content Negotiation**". L'algorithme de sélection peut se trouver sur le serveur ("server-driven negotiation") ou sur le client ("agent-driven negotiation").

5. Une visualisation du trafic réseau

5.1 Une communication en 10 paquets TCP

Pour être concret, supposons vouloir charger une page HTML nommée home-ferme-basique.html qui se trouve sur une machine serveur Myriam (192.168.2.1) faisant tourner un serveur Web Tomcat. En fait, Tomcat n'est pas seulement un serveur WEB, mais nous l'utiliserons tel quel ici, sur le port 8080. Sur la ligne d'URL du browser, on écrit :

<http://myriam:8080/home-ferme-basique.html>

En utilisant un sniffer comme EthetDetect :

x	#	Start Time	Client (IP:Port)	Server (IP:Port)	Protocol	Packets	Last T
	0	15:06:13.497	192.168.2.2:3133	192.168.2.1:8080	TCP	10	15:06

on peut constater immédiatement que :

- ◆ le protocole TCP, avec ses états, est bien sous-jacent; le début de la requête est bien une connexion TCP - un *three-way handshake*;
- ◆ la transaction HTTP réclame 3 paquets TCP;
- ◆ la connexion TCP est ensuite fermée – un *four-way handshake*.

connexion TCP (3)	packets in Connection No.0	Time Offset	Packet Len	Data ...	Data
		➡ 15:06:13.497	62	0	
		➡ 15:06:13.497	62	0	
		➡ 15:06:13.497	54	0	
		➡ 15:06:13.497	660	606	GET /home-ferme-basique.html HTTP/1.1 Host: myr
transaction HTTP (3)		➡ 15:06:13.497	146	92	HTTP/1.1 304 Non Modifié Date: Thu, 22 Jul 2004
		➡ 15:06:13.497	54	0	
		➡ 15:06:33.794	60	0	
		➡ 15:06:33.794	54	0	
déconnexion TCP (4)		➡ 15:06:40.903	54	0	

⁵ on distingue encore parfois le "proxy transparent" qui ne modifie pas la requête d'un "proxy non-transparent" qui modifie la requête pour ajouter des services supplémentaires à la réponse renvoyée.

5.2 La requête HTTP

Le premier paquet comportant des données représente bien la requête – à remarquer le flag PSH de TCP qui est utilisé :

Identifier: 17337	0000 00 60 97 70 46 7D 00 05 5D 2D C6 B0 08 00 45 00	. . . pF}...].-....E.
Offset: 16384	0010 02 86 43 B9 40 00 80 06 2F 65 C0 A8 02 02 C0 A8	..C. @.../e.....
TTL: 128	0020 02 01 DC 3D 1F 90 A6 D1 B6 94 00 F1 23 40 50 18	... =#OP.
Protocol: 6	0030 44 70 E9 B3 00 00 47 45 54 20 2F 68 6F 6D 65 2D	Dp.... GET /home-
Check Sum: 12133	0040 66 65 72 6D 65 2D 62 61 73 69 71 75 65 2E 68 74	ferme-basique.ht
Src IP: 192.168.2.2	0050 6D 6C 20 48 54 54 50 2F 31 2E 31 OD 0A 48 6F 73	ml HTTP/1.1..Hos
Dst IP: 192.168.2.1	0060 74 3A 20 6D 79 72 69 61 6D 3A 38 30 38 0D 0A	t: myriam:8080..
TCP Header	0070 55 73 65 72 2D 41 67 65 6E 74 3A 20 4D 6F 7A 69	User-Agent: Mozi
Src Port: 3133	0080 6C 6C 61 2F 35 2E 30 20 28 57 69 6E 64 6F 77 73	11a/5.0 (Windows
Dst Port: 8080	0090 3B 20 55 3B 20 57 69 6E 64 6F 77 73 20 4E 54 20	; U; Windows NT
Sequence: 2798761620	00A0 35 2E 31 3B 20 65 6E 2D 55 53 3B 20 72 76 3A 31	5.1; en-US; rv:1
Acknowledgement: 15803200	00B0 2E 30 2E 31 29 20 47 65 63 6B 6F 2F 32 30 30 32	.0.1) Gecko/2002
Head Len: 5	00C0 30 38 32 33 20 4E 65 74 73 63 61 70 65 2F 37 2E	0823 Netscape/7.
Flag: 0x18	00D0 30 0D 0A 41 63 63 65 70 74 3A 20 74 65 78 74 2F	0..Accept: text/
FIN: 0	00E0 78 6D 6C 2C 61 70 70 6C 69 63 61 74 69 6F 6E 2F	xml,application/
SYN: 0	00F0 78 6D 6C 2C 61 70 70 6C 69 63 61 74 69 6F 6E 2F	xml,application/
RST: 0	0100 78 68 74 6D 6C 2B 78 6D 6C 2C 74 65 78 74 2F 68	xhtml+xml,text/h
PSH: 1	0110 74 6D 6C 3B 71 3D 30 2E 39 2C 74 65 78 74 2F 70	tml;q=0.9;text/p
ACK: 1	0120 6C 61 69 6E 3B 71 3D 30 2E 38 2C 76 69 64 65 6F	lain;q=0.8;video
URG: 0	0130 2F 78 2D 6D 6E 67 2C 69 6D 61 67 65 2F 70 6E 67	/x-mng,image/png
Window: 17520	0140 2C 69 6D 61 67 65 2F 6A 70 65 67 2C 69 6D 61 67	,image/jpeg,imag
Check Sum: 59827	0150 65 2F 67 69 66 3B 71 3D 30 2E 32 2C 74 65 78 74	e/gif;q=0.2;text
Urgent Point: 0	0160 2F 63 73 73 2C 2A 2F 2A 3B 71 3D 30 2E 31 0D 0A	/css,*/*;q=0.1..
	0170 41 63 63 65 70 74 2D 4C 61 6E 67 75 61 67 65 3A	Accept-Language:
	0180 20 65 6E 2D 75 73 2C 20 65 6E 3B 71 3D 30 2E 35	en-us, en;q=0.5

c'est-à-dire de manière plus compréhensible :

```
GET /home-ferme-basique.html HTTP/1.1
Host: myriam:8080
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.1) Gecko/20020823 Netscape/7.0
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,video/x-
mng,image/png,image/jpeg,image/gif;q=0.2,text/css,*/*;q=0.1
Accept-Language: en-us, en;q=0.50
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Connection: keep-alive
If-Modified-Since: Thu, 22 Jul 2004 10:34:30 GMT
If-None-Match: W/"415-1090492470000"
Cache-Control: max-age=0
```

Packet Data

5.3 La réponse HTTP

Le paquet suivant renvoie bien la réponse tout en acquittant la requête :

TCP Header	0000 00 05 5D 2D C6 B0 00 60 97 70 46 7D 08 00 45 00	. . . pF}...].-....E.
Src Port: 8080	0010 02 A6 54 41 40 00 80 06 1E BD C0 A8 02 01 C0 A8	..TA@.....
Dst Port: 3134	0020 02 02 1F 90 0C 3E 01 19 4E 88 CA AF 42 29 50 18	... =#OP.
Sequence: 18435720	0030 20 4C 07 D3 00 00 48 54 54 50 2F 31 2E 31 20 32	L....HTTP/1.1 2
Acknowledgement: 3400483369	0040 30 30 20 4F 4B 0D 0A 45 54 61 67 3A 20 57 2F 22	00 OK..ETag: W/"
Head Len: 5	0050 34 31 35 2D 31 30 39 30 34 39 32 34 37 30 30 30	415-109049247000
Flag: 0x18	0060 30 22 0D 0A 4C 61 73 74 2D 4D 6F 64 69 66 69 65	0".."Last-Modifie
FIN: 0	0070 64 3A 20 54 68 75 2C 20 32 32 20 4A 75 6C 20 32	d: Thu, 22 Jul 2
SYN: 0	0080 30 30 34 20 31 30 3A 33 34 3A 33 30 20 47 4D 54	004 10:34:30 GMT
RST: 0	0090 0D 0A 43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20	..Content-Type:
PSH: 1	00A0 74 65 78 74 2F 68 74 6D 6C 3B 63 68 61 72 73 65	text/html;charse
ACK: 1	00B0 74 3D 49 53 4F 2D 38 35 39 2D 31 0D 0A 43 6F	t=ISO-8859-1..Co
URG: 0	00C0 6E 74 65 6E 74 2D 4C 65 6E 67 74 68 3A 20 34 31	ntent-Length: 41
Window: 8268	00D0 35 0D 0A 44 61 74 65 3A 20 54 68 75 2C 20 32 32	5..Date: Thu, 22
Check Sum: 2003	00E0 20 4A 75 6C 20 32 30 34 20 31 33 3A 35 39 3A	Jul 2004 13:59:
Urgent Point: 0	00F0 34 32 20 47 4D 54 0D 0A 53 65 72 76 65 72 3A 20	42 GMT..Server:
	0100 41 70 61 63 68 65 2D 43 6F 79 6F 74 65 2F 31 2E	Apache-Coyote/1.

soit en clair :

```
HTTP/1.1 200 OK
ETag: W/"415-1090492470000"
Last-Modified: Thu, 22 Jul 2004 10:34:30 GMT
Content-Type: text/html; charset=ISO-8859-1
Content-Length: 415
Date: Thu, 22 Jul 2004 13:59:42 GMT
Server: Apache-Coyote/1.1

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>Home-page de la ferme</title>

    <meta http-equiv="content-type"
        content="text/html; charset=ISO-8859-1">
</head>
<body>
<!-- Version 1.0 - Rédigé par super-génie -->
<h3>La ferme "G    on et Marie"</h3>

<h1>Une ferme mod    le ..</h1>

<h6>Bienvenue</h6>

</body>
</html>
```

6. La forme g    ale des messages du protocole HTTP

Un message HTTP⁶ sera toujours un requ  te du client au serveur ou une r  ponse du serveur au client. Dans les deux cas, et comme on peut le voir dans l'exemple ci-dessus, il respecte la structure suivante :

```
<message HTTP> = <ligne de d  marrage ou "start-line">
    <header g    ral du message>
    CR LF
    [<corps du message>]
```

Le header comporte lui-m  me un certain nombre de champs (  ventuellement, aucun). Chaque champ suit la syntaxe simple :

<nom du champ> : <valeur du champ> CR LF

Certains champs, appartenant  un **header g    ral**, peuvent se trouver aussi bien dans un message de requ  te du client que dans un message de r  ponse du serveur. Deux exemples :

Cache-Control: <description de l'utilisation du cache ; par ex. : no-cache>
--

Date: Tue, 3 July 2001 11:15:52 GMT
--

⁶ voir une description exhaustive sur <http://www.ietf.org/rfc/rfc2616.txt>

Les headers généraux sont, de manière exhaustive :

Cache-Control	Définit les directives de l'utilisation des caches, tant pour une requête que pour une réponse – valeurs classiques : no-cache, no-store
Connection	Dans une requête, l'émetteur peut configurer une connexion en exigeant que l'information ne soit pas communiquée par les proxies éventuels pour des connexions ultérieures. Dans une réponse, une valeur classique est "close", spécifiant que la connexion sera fermée après l'envoi de la réponse.
Date	Date-heure du message
Pragma	Pour des directives spécifiques à une implémentation (comme #pragma en C), valables aussi bien en requête qu'en réponse.
Trailer	Pour indiquer que certains headers se trouvent en fin de message – ne s'applique pas à Transfer-encoding et Content-length.
Transfer-Encoding	Indique si des transformations, dues à des changements de schémas d'encodage, ont été appliquées au message dans le but de transférer celui-ci de manière sûre. Valeurs possibles : "chunked" (découpage en morceaux de taille indiquée), "identity", "gzip", "compress", "deflate".
Upgrade	Permet à l'émetteur de signaler qu'il peut utiliser d'autres protocoles de communications et que le serveur peut les utiliser si cela l'intéresse. Exemple : Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11
Via	Permet aux proxies et passerelles de préciser les protocoles intermédiaires utilisés entre les proxies et les passerelles (utile avec TRACE). Exemple : Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1)
Warning	Avertissement sur d'autres transformations du message pour lesquelles il n'existe pas de header spécifique.

7. La requête

7.1 La forme et les méthodes

Elle a pour forme plus particulière :

```
<requête> =  <méthode> SP <URI de la ressource considérée> SP <version HTTP>
              <header général> | <header de requête> | <header d'entité>
              CR LF
              [<corps du message>]
```

La méthode peut être "POST", "GET", "OPTIONS", "PUT", etc. Plus précisément :

GET : Cette méthode permet de retrouver une information de nature quelconque identifiée par son URI. Dans le cas particulier où la ressource est en fait un service, ce sont les données produites par l'exécution de ce service qui constitueront la réponse.

HEAD : Comme la précédente, mais seul l'en-tête de l'URI visé constituera la réponse. L'objectif est clairement de connaître seulement les valeurs des champs, notamment de date de dernière mise à jour.

POST : Il s'agit ici de demander que le serveur accepte les données contenues dans le message comme constituants d'une requête pour la ressource-service dont l'URI est précisée. Un exemple classique et courant d'utilisation de cette méthode est l'envoi des données d'un formulaire pour une recherche dans une base de données. Ces données constituent alors ce que l'on appelle *l'entité de requête*.

PUT : Comme la précédente, cette requête demande que le serveur accepte les données contenues dans le message. Mais, cette fois, ces données seront placées telles quelles à l'URI considérée.

DELETE : A votre avis ?

TRACE : Il s'agit d'une méthode de débogage car son but est de permettre au client de voir ce qui a été reçu par sa cible, avec en particulier la possibilité de voir le cheminement (header *Via*).

OPTIONS : Cette méthode représente en fait une requête demandant les options disponibles pour la communication visant la ressource désignée par l'URI faisant partie du message.

La localisation de la ressource est évidemment vitale. A ce propos, il faut remarquer que :

- ♦ l'URI est classiquement un **URI absolu** (comme `http://www.prov-liege.be/vilvens.html`) ou, cas particulier, simplement "*", cette dernière possibilité signifiant que l'on ne vise pas une ressource particulière mais le serveur lui-même.

ex: GET `http://www.prov-liege.be/vilvens.html HTTP/1.1`

- si l'URI est **relatif**, il faut alors transmettre un champ complémentaire précisant l'emplacement réseau de la ressource visée. Par exemple :

`GET vilvens.html HTTP/1.1`

`Host : www.prov-liege.be`

La ligne complémentaire constitue *un champ du header de requête*. "Host" n'est pas le seul mot réservé possible : citons aussi "From", "Accept", "Expect", "User-agent", etc.

7.2 Quelques champs du header de requête

C'est dans le header de requête que le client peut transmettre des informations le concernant ou concernant sa requête.

1) Un champ classique est celui qui permet de préciser l'e-mail de l'utilisateur humain qui contrôle le user-agent qui a généré la requête. Son format est simplement :

From: <adresse e-mail>

2) Un autre exemple classique est précisément le champ "**Accept**". Ce champ de header de requête permet de *spécifier les types de media qui pourront être utilisés dans la réponse*, évitant ainsi une réponse incompréhensible par le client qui a formulé la requête. Le format d'un tel type est :

Accept: <type de media> / <sous-type de media ou "*"> [; q=<degré de préférence>] [,<type de media> / <sous-type de media ou "*"> [; q=<degré de préférence>]]

Le paramètre d'acceptation **q** (comme "quality") permet de spécifier ses préférences sur une échelle allant de 0 ("s'il n'y a vraiment rien d'autre") à 1 ("mon préféré entre tous"). Ainsi, Accept: text/plain; q=0.5, text/html, text/x-dvi; q=0.8

signifie "je préfère la version html, mais si elle n'est pas possible, alors je préfère la version en x-dvi. A remarquer que l'absence de clause q signifie implicitement que q vaut 1. Comme on peut aussi utiliser comme sous-type le symbole générique "*", on peut aussi écrire :

Accept: text/*, text/html, text/plain; q=0.5

Il n'y a pas, contrairement aux apparences, d'ambiguïté : c'est en effet la référence la plus spécifique qui est prioritaire, donc ici "text/html".

3) Des champs de header similaires, qui utilisent également le paramètre q, sont :

Accept-Language: <langue utilisée – par exemple "fr" ou "en">
Accept-Encoding: <méthode de codage – par exemple, "gzip", "compress" ou "deflate">
Accept-Charset: <jeu de caractères utilisé – par exemple, "iso-8859-5" ou "unicode-1-1">

4) Un dernier exemple est encore le champ User-Agent :

User-Agent: <nom du logiciel utilisé>
--

Par exemple :

User-Agent : CERN-LineMode/2.15

Ce champ fournit donc une information software sur le user-agent qui a générée la requête.

Le header de requête peut se résumer à un header général (c'est-à-dire non spécifique à une requête) comme "Cache-Control", "Date", etc.

7.3 La liste des headers de requête

Accept	Spécifie les types de media qui pourront être utilisés dans la réponse
Accept-Charset	Spécifie les jeux de caractères (charsets) qui pourront être utilisés dans la réponse
Accept-Encoding	Spécifie les schemes d'encodage qui pourront être utilisés dans la réponse (ne pas confondre avec les charsets)
Accept-Language	Spécifie les langues qui pourront être utilisées dans la réponse (par exemple : fr, en, en-gb, da, ...)
Authorization	Fournit l'information d'autorisation (les "credentials") qui seront utilisés par un gestionnaire d'accès côté serveur, comme par exemple un Realm de Tomcat ou Glassfish. Il peut s'agir par exemple d'un digest.

Expect	Pour demander une action particulière du serveur
From	Adresse e-mail de l'utilisateur qui se sert du user-agent.
Host	Pour spécifier l'hôte à qui s'adresse une requête GET – utile dans le cas d'hôtes virtuels sur la même adresse IP. Exemple : pour la ressource http://www.w3.org/pub/WWW/ : GET /pub/WWW/ HTTP/1.1 Host: www.w3.org
If-Match If-None-Match	L'objet de la requête est une entité et ne sera envoyé que si il correspond au(x) tag(s) d'entité spécifié –sinon, le client sera simplement informé de ce fait. Exemple : If-Match: "xyzzy", "r2d2xxxx", "c3piozzzz" OU le contraire
If-Modified-Since If-Unmodified-Since	L'objet de la requête ne sera envoyé que si il a été modifié depuis la date indiquée –sinon, le client sera simplement informé de ce fait. Exemple : If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT OU le contraire
If-Range	Dans le cas où l'on ne dispose plus que d'une partie d'une entité en cache, pour spécifier de n'envoyer le morceau manquant si il n'y a pas eu de changement; le tout sinon.
Max-Forwards	Pour limiter le nombre de proxies ou de passerelles qui peuvent relayer la requête (à utiliser avec les méthodes TRACE et OPTIONS).
Proxy-Authorization	Similaire à Authorization (c'est-à-dire fournit l'information d'autorisation - les "credentials") mais pour un proxy qui l'exige.
Range	Pour ne demander qu'un certain nombre de bytes. Exemple : Range: bytes=0-499
Referer	Permet de spécifier à partie de quelle Uri (quelle page) l'URI d'une méthode GET ou autre a été obtenu – très utile pour étudier une politique de référencement.
TE	Pour spécifier quel transfer-encoding le client souhaite pour sa réponse; il aurait pu s'appeler Accept-Transfer-Encoding
User-Agent	Nom du logiciel type browser utilisé.

8. La réponse

8.1 La forme et les méthodes

La réponse du serveur a pour forme plus particulière :

```
<réponse> = <version HTTP> SP <code> SP <texte information>
           <header général> | <header de réponse> | <header d'entité>
           CR LF
           [<corps du message – par exemple, la page HTML demandée>]
```

Il est prévu dans la définition du protocole que tout envoyer de message y précise la version qu'il est en état de recevoir. Cette version s'indique dans un champ situé en première ligne du message selon la syntaxe :

"**HTTP**/" <numéro principal>.<numéro secondaire>

Le code envoyé par le serveur est un entier à 3 chiffres lui permettant de signifier dans quelle mesure il peut donner suite à la requête reçue. Ces codes sont répartis en 5 groupes :

1xx	simple information : le traitement de la requête se poursuit ex: 100 ⇒ "Continue" : ce que le serveur a reçu constitue pour lui le début d'une requête; à ce stade, aucune erreur n'a été détectée, mais le client devrait poursuivre.
2xx	succès : la requête a été comprise et acceptée – l'affaire suit son cours ex : 200 ⇒ "OK" – ce qui a été demandé est envoyé 201 ⇒ "Created" : la ressource a été créée avec l'URI indiquée pour la référencer 202 ⇒ "Accepted" : la requête a été admise, mais non encore satisfaire
3xx	redirection : une autre action doit être entreprise par le user-agent pour compléter la réponse ex: 302 ⇒ "Found" : la ressource a été trouvée, mais avec une autre URI 305 ⇒ "Use Proxy" : la ressource visée doit être accédée au moyen d'un proxy dont le champ <i>Location</i> de la réponse fournit l'URI
4xx	erreur du côté client : soit dans la syntaxe, soit dans la nature même de la demande ex: 400 ⇒ "Bad request" : requête incompréhensible pour le serveur 401 ⇒ "Unauthorized" : la ressource ne peut être accédée que si le client s'authentifie, ce qu'il fera dans une requête munie d'un champ <i>WWW-Authenticate</i> . 404 ⇒ "Not found" : c'est clair ... 405 ⇒ "Not allowed" : la méthode (GET, POST, etc) est incompatible avec la ressource visée 407 ⇒ "Proxy Authentication Required" : le client doit s'identifier auprès du proxy au moyen d'un champ <i>Proxy-Authenticate</i> . 415 ⇒ "Unsupported Media Type" : c'est clair ...
5xx	erreur du côté du serveur : la requête semblait fondée, mais le serveur n'a pas la satisfaire ex: 500 ⇒ "Internal error" : autrement dit, rien ne va plus ... 503 ⇒ "Service Unavailable" : le serveur ne peut satisfaire la requête pour l'instant (par exemple, parce que le serveur est en maintenance)

Les champs du header de réponse utiliseront un mot réservé comme *Location*, *Proxy-Authenticate*, etc. Enfin, les informations demandées, en cas de succès, constituent la fin du message : il peut s'agir des données d'un fichier ou des résultats d'un GCI ou d'une servlet. Ces données constituent l'*entité de réponse*.

8.2 Quelques champs du header de réponse

Dans ce header, le serveur peut fournir des informations sur lui-même ou sur la ressource fournie, notamment quant à l'accès ultérieur de celle-ci. Par exemple, on peut donner le nom du programme serveur qui traite la requête :

Server: <nom du programme, par ex. : CERN/3.0>

Il existe aussi des champs se référant à l'authentification, comme **Proxy-Authenticate** et **WWW-Authenticate**.

8.3 La liste des headers de réponse

Accept-Ranges	Le serveur peut (sans obligation) indiquer qu'il a accepté l'indication de Range de la requête ou pas.
Age	Fournit une estimation en secondes du temps écoulé depuis la génération de la réponse par le serveur.
ETag	Fournit une indication sur l'état d'une entité (son "tag")
Location	Précise que la ressource obtenue provient d'une autre URI (code 201) ou doit être redirigée (code 3xx).
Proxy-Authenticate	Fournit le challenge à utiliser pour réaliser l'authentification auprès d'un proxy (code 407).
Retry-After	Indique une estimation du temps (une date ou un intervalle en secondes) d'inaccessibilité du service demandé (code 503).
Server	Information sur le logiciel utilisé par le serveur.
Vary	Indique dans quelle mesure il fait usage d'un cache : ce champ fournit la liste des champs utilisés pour déterminer si l'usage du cache peut être fait ou pas.
WWW-Authenticate	Fournit le challenge à utiliser pour réaliser l'authentification auprès du serveur (code 401 : Unauthorized) – voir paragraphe 10.

9. Les entités

9.1 Une réponse qui encapsule une information

Une requête ou une réponse peut transporter une "entité", comportant elle-même un header (appelé header d'entité) et un corps éventuel. Un champ du header véhicule une métainformation à propos de l'entité ou de la ressource visée par l'URI du message (dans ce cas, l'entité n'a pas de corps). Par exemple, la réponse suivante comporte un header d'entité :

HTTP/1.0 503 Service Unavailable Server: Squid/2.1.PATCH1	Header de réponse
Mime-Version: 1.0 Date: Sun, 15 Jun 2008 06:57:09 GMT Content-Type: text/html Content-Length: 867 Expires: Sun, 15 Jun 2008 06:57:09 GMT	Header de l'entité
X-Squid-Error: ERR_DNS_FAIL 0 X-Cache: MISS from delorie.com Proxy-Connection: close	

On constate immédiatement que le premier header précise la version de MIME qui permettra de désigner le type des données constituant le corps du message, ce qui se fera par :

Content-Type : <codes MIME>

9.2 La liste des headers d'entité

Allow	Indique la seule méthode qui convient pour la ressource visée code 405)
-------	---

Content-Encoding	Equivalent des Accept pour une requête
Content-Language	
Content-Length	
Content-Location	
Content-Range	
Content-Type	
Content-MD5	Pour permettre un contrôle d'intégrité de l'entité transportée.
Last-Modified	Indique la date de la dernière modification de l'entité
Expires	Donne la date à partir de laquelle le cache ne sera plus utilisé pour cette entité

10. La sécurité HTTP

10.1 L'authentification basique

Nous avons déjà fait allusion au header Authorization. En effet, si un client se voit refuser l'accès à une ressource (par une requête GET par exemple), il obtiendra du serveur une réponse de ce type :

HTTP/1.1 401 Authorization Required
WWW-Authenticate: Basic realm="Elus"

Par "realm", il faut comprendre un domaine sur le serveur qui rassemble un certain nombre d'informations soumis aux mêmes règles de sécurité. Une sécurité élémentaire, implémentée dans HTTP, est celle qui est basée sur *la paire formée d'un nom d'utilisateur et d'un mot de passe*. Lorsqu'un user agent veut accéder à une ressource protégée, le serveur lui demandera d'introduire son nom et son mot de passe. Plus précisément, le header **Authorization** de la requête du client contiendra un champ de la forme :

BASIC nom:password

donc par exemple :

Authorization : Basic S1AJFNE8F7RBFKE7==

Ce champ est codé selon **l'encodage Base64**. Celui-ci décompose les groupes successifs de 24 bits en 4 groupes de 6 bits. Chacun de ces groupes est alors envoyé en tant que caractère ASCII, selon la convention que "A" correspond à 0, "B" à 1, ..., "Z" à 25, "a" à 26, ..., "0" à 52, "+" à 62 et "/" à 63. Comme la fin du message peut contenir des octets vides, ceux-ci sont remplis avec "==" et "=" (un tel remplissage est appelé un *padding*). Le serveur peut donc rechercher ces données dans un fichier qu'il gère et authentifier ou pas le client.

10.2 L'authentification par digest

Dans le cas d'un refus d'accès, le client peut aussi recevoir une réponse du type suivant :

HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest realm="Elus",
qop="auth",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
opaque="5ccc069c403ebaf9f0171e9517f40e41"

Cette fois, la réponse comporte un challenge (un "nonce" [*number used once*], c'est-à-dire un

paramètre variable au cours du temps) sur lequel le client va calculer une réponse basée sur des digests (des MD5) faisant intervenir ce challenge, un deuxième challenge (*cnonce* - le "client nonce") qu'il aura généré et son mot de passe (qui donc ne circulera pas sur le réseau) :

```
Authorization: Digest username="Dieu",
realm="Elus",
nonce="dcd98b7102dd2f0e8b11d0f600fb0c093",
uri="/admin/",
qop=auth,
nc=00000001,
cnonce="0a4f113b",
response="6629fae49393a05397450978507c4ef1",
opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Le paramètre *opaque* ne fait que désigner le realm et est en fait un cookie (voir plus loin). Le paramètre *qop* peut valoir "auth" ou "auth-int", ce qui change les formules utilisées pour la réponse : dans le deuxième cas, le contenu même de l'entité transportée intervient dans le calcul.

Le serveur, grâce au *username*, peut trouver dans ses fichier le mot de passe correspondant et calculer un digest local similaire : le fait que les deux digests soient égaux assure l'authentification.

10.3 HTTPS

On désigne par HTTPS la version sécurisée de HTTP. En fait, il s'agit du protocole HTTP utilisé sur une communication client-serveur utilisant le protocole **SSL** (Secure Socket Layer), protocole visant au cryptage des données transmises et se situant entre la couche transport (soit TCP) et le protocole applicatif utilisé (ici, HTTP). Donc, en pratique, il suffit de remplacer "http" par "https" dans les URLs concernés.

Techniquement, pour le lecteur au fait des éléments de cryptographie (mais voir "Java IV" pour une description détaillée), voici comment les choses se passent :

- ◆ les systèmes communicants commencent par s'identifier auprès des autres au moyen de certificats;
- ◆ ils négocient une clé asymétrique;
- ◆ celle-ci permet alors la négociation d'une clé symétrique;
- ◆ cette clé est ensuite utilisée pour le reste de la communication (c'est la clé de session).

11. Les cookies

On l'a bien précisé, HTTP est un protocole est **sans état** : une fois une transaction client-serveur terminée, il ne reste aucune trace de l'identification de l'utilisateur, si bien qu'il n'est pas possible, par exemple, de gérer une information (par exemple, un compteur, une somme, une date) qui conserverait sa valeur entre deux dialogues. Autrement dit, il n'est pas possible de gérer une "session HTTP". On conçoit sans peine le problème que cela peut causer quand il s'agit d'un site de e-commerce encombré des caddies virtuels de multiples utilisateurs !

Pour cette raison, deux nouveaux headers ont été ajoutés au protocole de base afin de permettre la gestion de "**cookies**". Un **cookie** est une information (plus précisément un fichier texte) envoyée à un browser par un serveur qui vient d'être accédé afin que cette information

soit sauvegardée sur le client. Chaque fois que le browser du client s'adressera à ce serveur, il lui renverra le contenu du cookie. De cette manière, un serveur peut, par exemple, identifier d'une manière unique un client en lui envoyant un identifiant dans un cookie, identifiant valide puisque c'est le serveur lui-même qui a créé l'identification qu'il récupérera. Le cookie peut aussi servir, par exemple, à mémoriser le nombre de visite sur un site, la date de la dernière visite, une liste de paramètres de préférences, des informations à conserver d'une page à une autre (on fabrique ainsi une **session**), etc.

Le texte d'un cookie possède une structure bien définie :

- ◆ un nom à qui sera associée la valeur du cookie;
- ◆ une date d'expiration permettant de limiter la durée de vie (Max-Age) du cookie (par défaut, le cookie est détruit à la fermeture du browser);
- ◆ le chemin (Path) d'origine du cookie;
- ◆ un nom de domaine permettant d'identifier le cookie parmi tous ceux stockés sur la machine cliente;
- ◆ une variable secure indiquant si l'accès au cookie est protégé.

Pour que l'utilisation des cookies soit possible, deux nouveaux headers ont été ajoutés : Cookie et Set-Cookie. Ces nouveaux headers sont définis dans la RFC 2109. Il devient alors possible de gérer une session entre un client et un serveur, session de durée bien déterminée et relativement courte. Cette session pourra être clôturée par l'un des deux intervenants.

En pratique, supposons qu'un user-agent contacte un serveur HTTP en remplissant un formulaire HTML pour se connecter. Sa requête a un aspect du type suivant :

```
POST /machin/login HTTP/1.1
User-Agent: Mozilla/4.0
Accept: text/* */
CR
<infos du formulaire : ?nom=Genius&prenom=James&pwd=tropsucces>
```

Le serveur initialise alors une session en retournant dans sa réponse un header Set-Cookie complémentaire :

```
HTTP/1.1 200 OK
Date: Saturday, 23-January-99 10:25:12 GMT
Server: Netscape Server/3.55
MIME-version: 1.0
Content-type: text/html
Content-length: 2530
Set-Cookie: leclient="Genius_J_trop"; Max-Age=3600; Version="1"; Path="/machin"
```

En réalité, le header Set-Cookie peut donc comporter des clauses supplémentaires comportant des indications sur le groupe d'URLs pour lequel le cookie est utilisable (Path – optionnel), le domaine, ... Ici, une autre valeur additionnelle est la durée de vie du Cookie, exprimée en secondes dans la clause Max-Age (un temps nul signifie que le cookie est supprimé). A remarquer que la clause Version est obligatoire, mais vaut toujours 1 pour l'instant ...

Notre client commence à acheter des articles choisis dans des formulaires. A chaque POST qu'il effectuera, il renverra au serveur le(s) cookie(s) fourni par la réponse précédente du serveur, flanqué des attributs \$Version et \$Path coorespondants à ceux précisés par le serveur. Cela donne par exemple :

```
POST /machin/trade HTTP/1.1
User-Agent: Mozilla/4.0
Accept: text/* */
Cookie: $Version="1"; leclient="Genius_J_trop"; $Path="/machin"
CR
<infos du formulaire : ?article=F02102&libelle=grande_bouteille_val_dieu>
```

Le serveur répond :

```
HTTP/1.1 200 OK
...
Set-Cookie: larticle="val_dieu_2"; Version="1"; Path="/machin"
```

Imaginons que le client choisisse à présent le mode de paiement dans un nouveau formulaire :

```
POST /machin/pay HTTP/1.1
User-Agent: Mozilla/4.0
Accept: text/* */
Cookie: $Version="1"; leclient="Genius_J_trop"; $Path="/machin"
$Version="1"; larticle="val_dieu_2"; $Path="/machin"
CR
<infos du formulaire : ?moyen=visa&numero=6900769>
```

Le serveur envoie un nouveau cookie :

```
HTTP/1.1 200 OK
...
Set-Cookie: lemoyen="visa"; Version="1"; Path="/machin"
```

Si le client choisit en définitive de commander :

```
POST /machin/execute HTTP/1.1
User-Agent: Mozilla/4.0
Accept: text/* */
Cookie: $Version="1"; leclient="Genius_J_trop"; $Path="/machin"
$Version="1"; larticle="val_dieu_2"; $Path="/machin"
$Version="1"; lemoyen="visa"; $Path="/machin"
CR
<infos du formulaire : ?choix="yes">
```

Le serveur conclut :

```
HTTP/1.1 200 OK
leclient="Genius_J_trop"; Max-Age=0; Version="1"
```

et la transaction est terminée.



Nous pouvons voir passer dans la réponse du serveur des balises <HTML> et </HTML> qui limitent une page HTML. Le moment est sans doute venu de comprendre la structure d'un tel document.

II. Le langage HTML : les bases



Qui veut trop trouver ne trouve rien.

(H. de Montherlant, Carnets)

1. Le langage HTML

Le terme **HTML** est un acronyme provenant de **HyperText Markup Language** - "langage de balisage hypertexte" en français. Il a donc pour but, comme son nom l'indique, de décrire un document hypertexte au moyen de **balises** ou **marqueurs** (en anglais et en franglais, on dit des "**tags**"). En fait, il constitue un sous-ensemble d'un langage plus général appelé SGML (Standard Generalized Markup Language). Tous deux ont pour buts de décrire le contenu d'un document, mais pas sa mise en page. En effet, le document devra être visualisé sur des plates-formes très différentes, si bien que l'on ne peut préjuger de l'aspect final du document. Ceci dit, sous la pression des utilisateurs, le langage s'est enrichi de balises qui concernent bel et bien la présentation ...

Les documents HTML présentent deux qualités qui justifient leur utilisation intensive sur Internet :

- ◆ les documents HTML sont généralement peu volumineux; leur transfert occasionne donc un minimum de trafic réseau;
- ◆ les documents HTML sont multi plates-formes et indépendants du matériel utilisé.

La norme actuelle est passée de HTML 3.2 à **HTML 4.01**. On peut trouver sa spécification sur <http://www.w3.org/TR/REC-html40>. Une extension actuelle est le **XHTML**, reformulation structurée de HTML 4 mais plus rigoureuse (toute balise ouvrante réclame sa balise fermante). Mais le futur est sans doute HTML 5, bien adapté aux nouvelles technologies et revenant à la permissivité de base qui a justifié le succès de HTML ...

2. Les balises, les marqueurs, les tags

Les pages WEB actuelles sont donc conçues au moyen de symboles prédéfinis appelés des marqueurs ou des balises.

Le principe des **balises** (ont dit encore "marqueurs" ou surtout "tags") est assez semblable, par écrit, à ce qu'une secrétaire peut recevoir comme directives de son patron au moyen d'un dictaphone. Ainsi, si le message est :

« Mon nom en grand, centré »
« Retrait » « Mon amour, c'est le cœur brisé que je dois t'annoncer notre rupture (souligner rupture) » « A la ligne » « Trop de choses nous séparent » « A la ligne » « en grand : ADIEU » « signé : Alberto (en penché) »

on peut considérer que le texte à taper est orné de marqueurs comme « A la ligne », « Souligner », etc. La description du document est donc :

```
<Centré><Grand> Alberto Macho <Fin grand><Fin centré>
<Retrait> Mon amour, c'est le cœur brisé que je dois t'annoncer notre <Souligner> rupture
<Fin souligner>
<A la ligne><Retrait> Trop de choses nous séparent
```

```
<A la ligne> <En grand> ADIEU <Fin en grand>
<A la ligne> <Penché>Alberto <Fin penché>
```

Le texte qui sera effectivement tapé aura l'aspect suivant :

```
Alberto Macho
Mon amour, c'est le coeur brisé que je dois t'annoncer notre rupture
Trop de choses nous séparent
ADIEU
Alberto
```

La description des pages WEB se pratique de la même manière, en utilisant le langage HTML. Il faut évidemment disposer d'un logiciel qui saura comprendre les balises HTML afin de réaliser la présentation écran. C'est l'un des rôles du browser.

De nombreuses balises HTML se comportent comme l'exemple

```
<Penché>Alberto <Fin penché>
```

Autrement dit, une balise indique le début d'une façon de faire et l'autre en indique la fin. En HTML, une balise marquant une fin commence par '/', ce qui donnerait ici (si la balise 'Penché' existait !) :

```
<Penché>Alberto </ Penché>
```

3. Structure de base d'un document HTML

Pour qu'un browser reconnaissse un document est un document HTML, il est souhaitable (et peut-être indispensable dans l'avenir) qu'il contienne 3 balises d'identification :

<HTML> : signale que le document est codé en HTML;
<HEAD> et </HEAD> : délimitent la partie en-tête; celle-ci se réduit en général à un titre et des commentaires et, de toute manière, ne contient pas de véritable texte de la page WEB;
<BODY> et </BODY> : contient l'essentiel de la page WEB.

La structure de base d'un document HTML est donc :

```
<HTML>
```

```
  <HEAD>
```

```
  </HEAD>
```

```
  <BODY>
```

```
  </BODY>
```

```
</HTML>
```

De plus, un document valide doit déclarer, en tête de document, la version d'HTML qu'il utilise par une ligne (en fait, de commentaires) de l'un des 3 types :

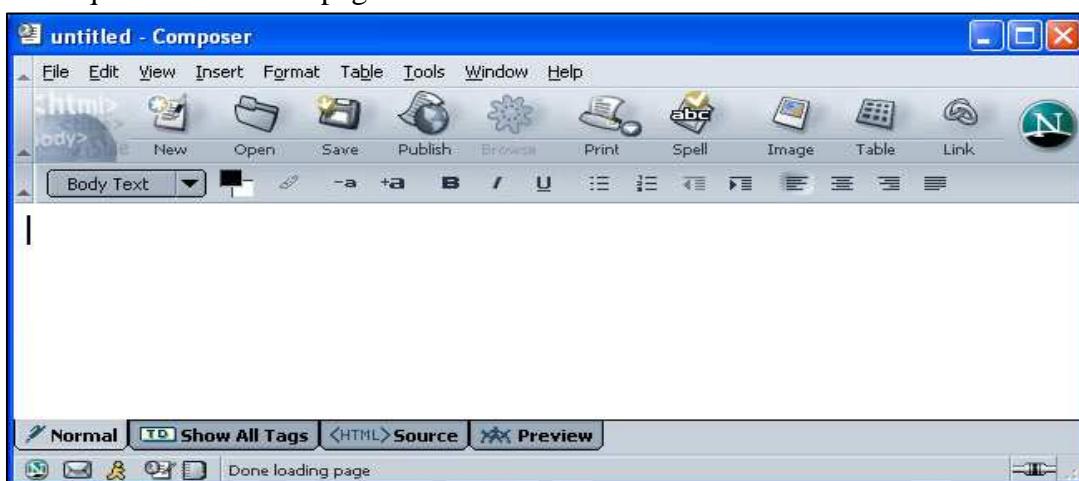
- ◆ `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 //EN">`
si le document HTML se conforme strictement à la norme 4.0;
- ◆ `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional //EN">`
si le document HTML se conforme à la norme 4.01 mais en admettant des tags qui seront éliminés dans les versions futures (*deprecated*);
- ◆ `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset //EN">`
si le document HTML est du type Transitional 4.0 avec utilisations de frames (document composés).

On parle encore de ***document type declaration*** (DTD)⁷. En complément, l'URL du document de référence peut être indiqué :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html40/strict.dtd">
```

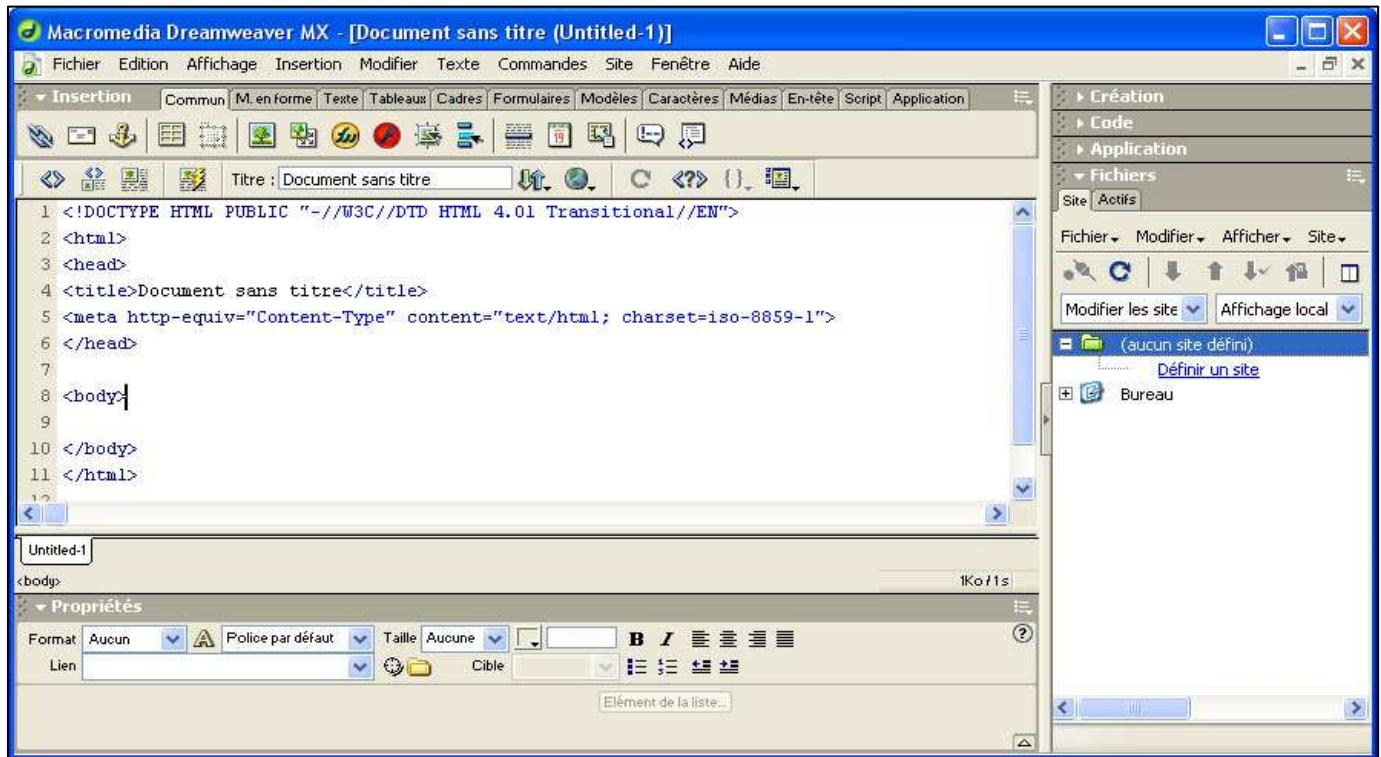
4. Création d'une page HTML

A priori, on crée une page HTML avec un éditeur de textes quelconque, vu la nature exclusivement textuelle d'une telle page. En pratique, évidemment, plutôt que de concevoir ses pages "au marteau et au burin", on préfère disposer d'un logiciel d'édition qui automatise un certain nombre de pages. Certains de ces éditeurs sont complexes (comme Microsoft FrontPage), mais surtout, certains d'entre eux créent des ***balises propriétaires***, c'est-à-dire uniquement reconnues par le browser qui fait partie de la même suite Internet. La raison incite donc à utiliser un éditeur de pages WEB qui respecte les standards internationaux. Même ainsi, le choix est encore assez varié; pour notre part, nous avons en son temps utilisé le "composeur" de **Netscape 7.*** (ici, une version anglaise) – mais à présent, c'est **Macromedia Dreamweaver MX** qui semble l'offre la plus intéressante actuellement – ce dernier réclame cependant un certain apprentissage qui dépasse le cadre de notre exposé. Le lancement de ce composeur de Netscape 7.* place automatiquement dans une page blanche :



⁷ on emploie le même terme en XML

tandis que Dreamweaver donne :



En fait, on dispose de différents onglets donnant des visions différentes de la même page WEB, dont les deux plus importants sont

- ◆ **le mode page** : on dispose à sa guise les éléments d'information (texte, image, lien,) sans que les tags HTML n'apparaissent; c'est l'onglet Normal de Netscape, obtenu par  sous Dreamweaver;
- ◆ **le code source** : on voit le source HTML proprement dit, donc ce qui a été généré automatiquement et ce que l'utilisateur a ajouté; libre à ce dernier de modifier ou d'ajouter des tags "à la main"; c'est l'onglet <HTML>Source de Netscape, obtenu par  sous Dreamweaver.

On peut vérifier qu'une structure minimale a été mise en place :

untitled.html
<pre> <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"> <html> <head> <title></title> <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1"> </head> <body>
 </body> </html> </pre>

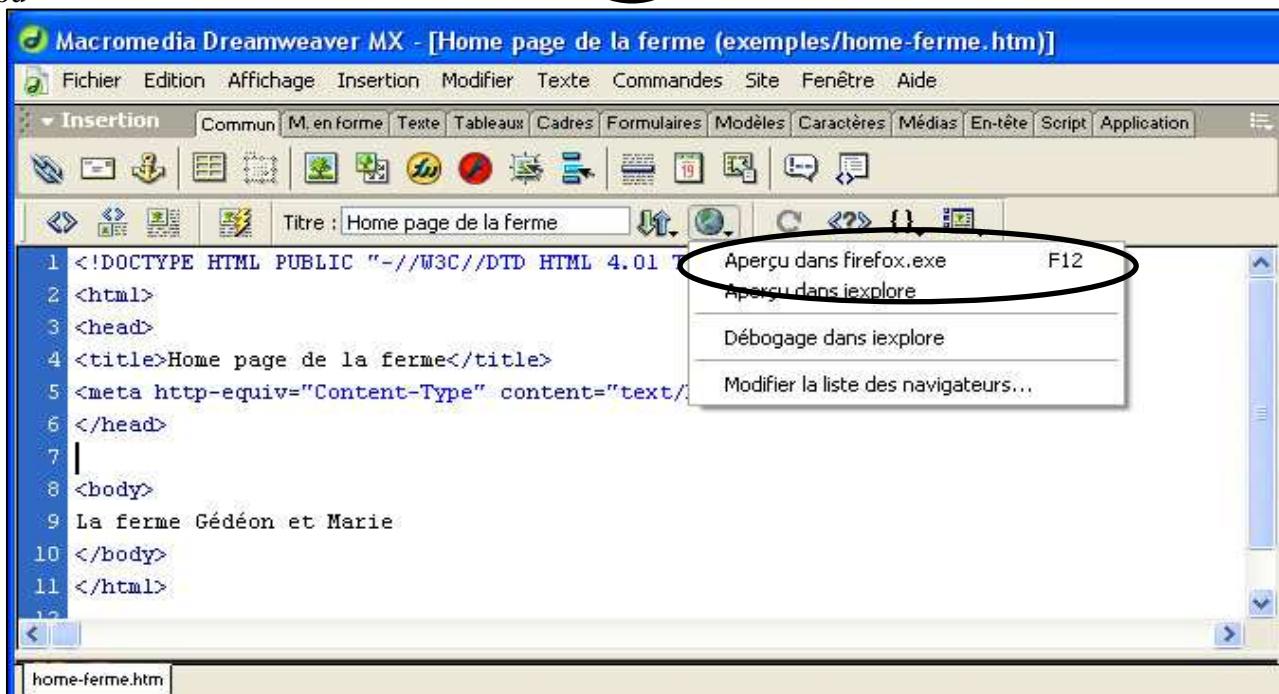
Il est aussi possible de créer une page à partir d'un template, personnel ou fourni par Netscape ou encore d'utiliser un assistant.

Il est à présent possible de taper du texte dans la page vide (par exemple : "La ferme Gédéon et Marie" – nous préparons le site WEB d'une ferme ;-)), puis de sauver l'œuvre sous un nom quelconque avec l'extension htm ou html (par exemple : home-ferme.html). Il est possible de définir le titre de la page, titre qui sera affiché lorsque la page sera accédée.

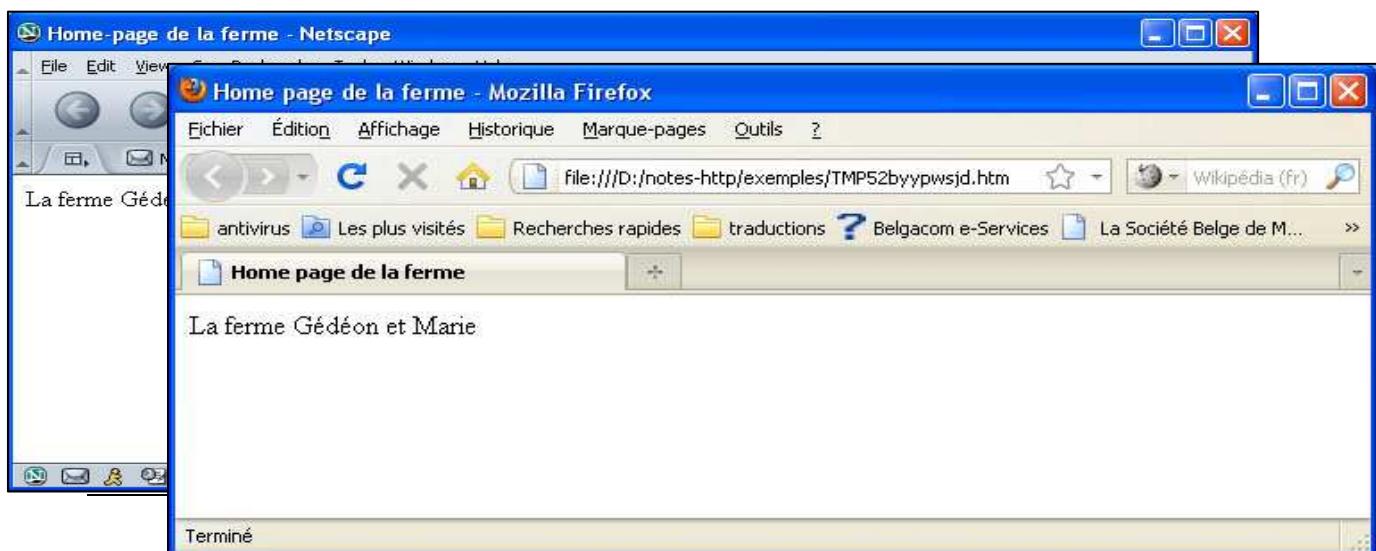
Par rapport à la structure de base, on peut constater dans le code HTML l'existence de balises complémentaires ajoutées par Netscape. La seule qui nous intéresse directement est la balise <TITLE> </TITLE> qui délimite la zone de titre de la page et qui comporte bien le titre que nous avons précisé. Rien ne nous empêche de modifier quelque chose dans ce source HTML, par exemple le titre. On peut alors visualiser le résultat des modifications en appelant le browser par :



ou



Après quoi le browser interprète notre splendide page :



Remarques

1) Comme évoqué à propos des versions, il est possible de placer des commentaires dans le source HTML. La norme veut qu'ils soient indiqués par

<!-- au début et
--> à la fin.

Ainsi,

<!-- Version 1.0 - Rédigé par super-génie -->

n'aura aucun effet. Certains browsers se contentent de <! et >. C'est le cas dans le source généré automatiquement par Netscape.

2) On peut créer un nouveau document depuis l'éditeur en cliquant sur l'icône :



5. Encoder du texte

Pour l'instant, notre page est quasiment vide. La première chose à faire est d'y placer un (ou plusieurs) titre(s). Encodons, par exemple, dans l'éditeur graphique :

La ferme Gédéon et Marie

Une ferme modèle ...

Bienvenue

On pourra vérifier que le source HTML confectionné est :

```
<P>La ferme &quot;G&acute;d&acute;on et Marie&quot;</P>  
<P>Une ferme mod&egrave;le ...</P>  
<P>Bienvenue</P>
```

Gloups ! Surprenant, n'est-ce pas ? Voici ce qui se passe ...

- ♦ Les balises **<P>** et **</P>** délimitent les paragraphes (**</P>** est en fait optionnel). On a effectivement frappé Enter après chaque ligne, ce qui signifie pour l'éditeur un changement de paragraphe. Si l'on souhaite un saut de ligne au sein d'un même paragraphe, il faut utiliser la balise **
** (visuellement, elle provoque d'ailleurs un écart moins important entre les lignes); elle peut s'obtenir par les touches combinées SHIFT+ENTER.
- ♦ Les caractères accentués (é, è, à, ...) sont propres à la langue de Molière et ne figurent pas dans le code ASCII à 7 bits. Or, HTML utilise justement celui-ci parce qu'il est le seul à être correctement compris par toutes les plates-formes. La plupart de celles-ci interprètent

cependant les caractères exotiques comme faisant partie de la norme **ISO-Latin-1** (c'est celle qui est utilisée par défaut sous Netscape - voir en 3.* Options-General preferences-Font). Dès lors, HTML a choisi pour le codage des caractères non standards un codage particulier connu comme utilisant *les entités de caractères*. Celles qui nous intéressent directement sont *les entités nominales* : il s'agit tout simplement d'un nom donné à chaque caractère spécial. Par exemple, ‘é’ est connu par ‘eacute’. Mais au sein d'un texte, il serait difficile de reconnaître le nom du caractère des caractères standards. Pour cette raison, un caractère particulier (‘&’) marque le début du nom et un autre (‘;’) en marque la fin. Donc, finalement,

- ◊ un ‘é’ se codera é
- ◊ un ‘à’ se codera à
- ◊ un ‘è’ se codera è
- ◊ etc.

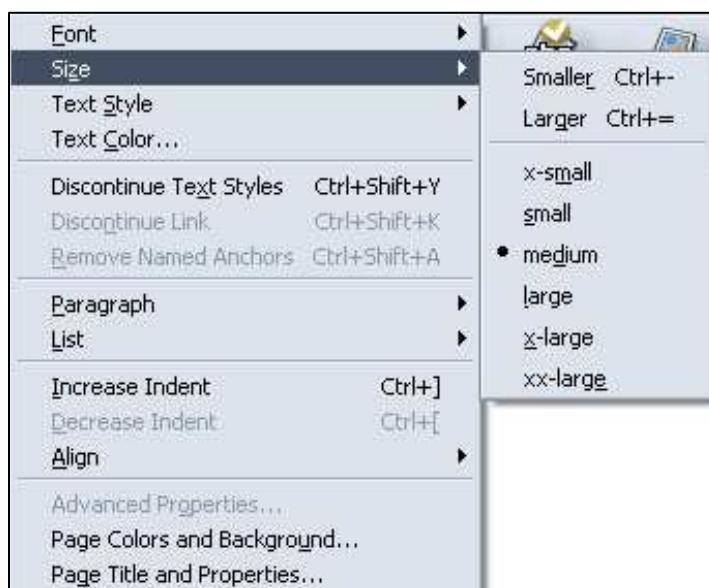
Fort heureusement, l'utilisateur de Netscape ne devra pas encoder les entités nominales : Netscape transformera de lui-même les caractères spéciaux en ces entités (ouf !).

6. Placer des titres

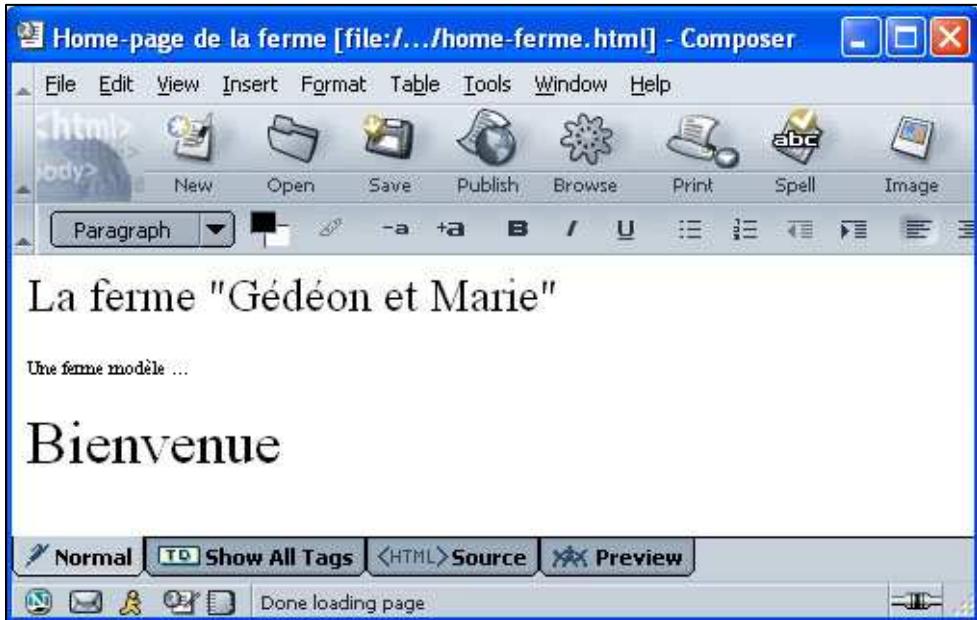
Dans l'esprit de leur créateur, les lignes encodées doivent servir de titre pour la home-page. Tels quelles, ces lignes sont un peu malingres. Netscape propose les outils permettant de modifier l'apparence d'un texte, d'une part dans la barre d'outils :



mais aussi dans le sous-menu correspondant à Format :



On peut ainsi obtenir par simple modification de la taille des caractères (successivement, x-large, x-small et xx-large) :



Les balises HTML créées à cette occasion sont visibles dans l'éditeur de texte :

```
<P><FONT SIZE=+2>La ferme &quot;G&acute;d&acute;on et arie&quot;</FONT></P>
<P><FONT SIZE=-2>Une ferme mod&egrave;le ..</FONT>.</P>
<P><FONT SIZE=+3>Bienvenue</FONT></P>
```

Peu importe, puisqu'elles sont créées automatiquement ...

On peut reprocher à ces balises de ne pas être forcément portables sur toutes les plates formes. Elles sont en tous cas reconnues par Netscape, Mozilla Firefox et Internet Explorer, soit 90% (au bas mot) des browsers utilisés.

Les balises reconnues à coup sûr sont les balises de titres **<H1></H1>** (le plus grand), **<H2></H2>**, ..., **<H6></H6>** (le plus petit), disponible dans la barre d'outils :



Ainsi, les titres évoqués ci-dessus peuvent devenir ici :

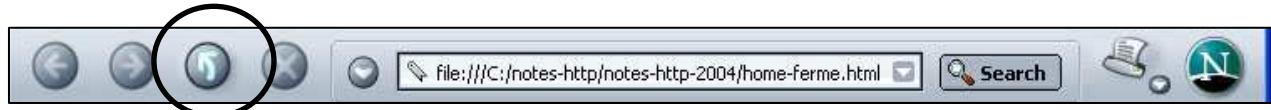
```
<P><H3>La ferme &quot;G&acute;d&acute;on et Marie&quot;</H3></P>
<P><H1>Une ferme mod&egrave;le ..</H1>.</P>
<P><H6>Bienvenue</H6></P>
```

L'aspect dans le browser sera le suivant :



Remarques

1) Il se peut qu'après une modification par l'éditeur, la visualisation ne tienne pas compte de ces derniers changements. Dans ce cas, il faut demander au browser de **recharger la page** en sollicitant le bouton Recharger [reload] :



2) L'onglet "Show all tags" donne :



Un double-clic sur un tag permet d'éditer toutes ses caractéristiques (comme ses attributs – voir plus loin).

7. D'autres possibilités de mises en forme

Il est possible :

- ♦ de mettre du texte en gras, en italique ou en souligné (mais cette dernière possibilité n'est pas intéressante à cause du format des liens hypertextes) :



- ♦ de donner des couleurs au texte en choisissant dans une palette :



- ♦ de reculer ou d'avancer la marge d'un texte :



- ♦ d'aligner à gauche, au centre, à droite :



On peut toujours faire apparaître par un **clic droit** un menu flottant permettant les couper-coller. L'ensemble des ces possibilités, complétées de bien d'autres, sont disponibles dans le menu à l'item principal Format.

8. Les propriétés

Toujours par le menu format, on peut fixer d'autres propriétés de la page, comme les couleurs, par

Format → Page Colors and Background



On peut ainsi obtenir finalement le résultat suivant dans le browser :



Le code HTML correspondant est :

home-ferme.html (1)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Home-page de la ferme</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
</head>

<body text="#000000" bgcolor="#33ccff" link="#000099" vlink="#990099"
alink="#000099">

<h2 align="center"><font color="#3333ff">La ferme "G&eacute;d&eacute;on et
Marie"</font></h2>

<h1><font color="#ff0000">Une ferme mod&egrave;le ...</font></h1>

<h2 align="center"><font color="#666600">Bienvenue</font></h2>
<br>

</body>
</html>
```

On peut remarquer que les couleurs sont codées selon les conventions **RGB**. Dans ce système de codage, une couleur est désignée par 24 bits; chaque byte correspond à une proportion de l'une des trois couleurs primaires (rouge-Red, vert-Green, bleu-Blue). Dans ces conditions, 0 0 0 est le code du noir et 255 255 255 est le code du blanc. Les valeurs sont codées en hexadécimal, ce qui est indiqué par le symbole '#'.

9. Les tags META

L'objectif de tout créateur de pages est que son œuvre soit lue par le maximum d'internautes. Encore faut-il que les dites pages soient trouvées et, pour cela, il faut que les **robots indexeurs** des moteurs de recherche puissent les référencer et obtenir des informations sur elles. C'est ici qu'intervient une balise un peu particulière, le tag <META>. Comme son nom l'indique, il ne sert pas à insérer des informations dans la page, mais à contenir des informations sur ces informations : on peut bel et bien parler de "**meta-information**".

La balise <META> ne se contente pas, forcément, d'être une simple bascule comme ou <BODY> : elle possède des attributs, c'est-à-dire champs complémentaires fournissant des indications diverses. Nous renconterons souvent de telles balises. Ici, la forme générale est :

```
<META NAME=nom de champ CONTENT=valeur du champ >
```

Par exemple, on peut imaginer :

```
<meta name="Author" content="Claude Vilvens">
```

En fait, le nom des champs n'est pas vraiment formalisé. Mais, vu le but poursuivi, il y a tout intérêt à utiliser les champs connus des moteurs de recherche, comme "author", "copyright", "description", "keywords", etc.

On peut aussi remarquer le tag :

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

qui fait explicitement référence au protocole HTTP. Il est donc possible de définir les propriétés du document par rapport à ce protocole, propriétés qui affecteront le header HTTP correspondant. Dans le même style, on peut par exemple ajouter :

```
<meta http-equiv="Expires" content="Tue, 20 Aug 1996 14:25:27 GMT">
```

La balise META admet des attributs complémentaires, par exemple :

```
<META name="Author" lang="fr" content="Albert Dugenou">
```

qui spécifie en plus la langue. Cependant, par rapport aux moteurs de recherche, c'est certainement l'attribut "**keywords**" qui présente le plus d'intérêt puisqu'elle permet de spécifier une liste de mots clés que de nombreux moteurs utiliseront. La syntaxe est simplement :

```
<META name="keywords" content="liste de mots séparés par une virgule">
```

Par exemple, on pourra placer :

```
<META NAME="KeyWords" CONTENT="Malacologie, Mollusques, Belgique">
```

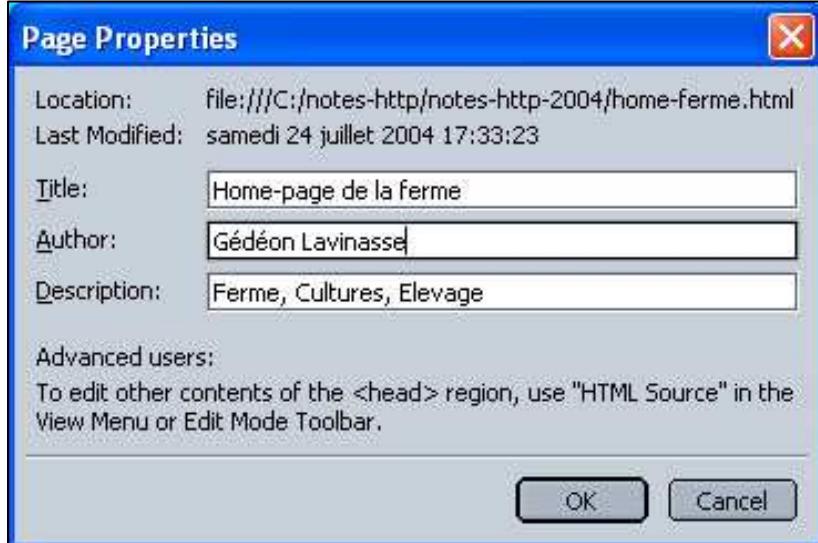
dans la home page du site d'une société savante qui étudie les Mollusques. Le champ content peut contenir 1000 caractères au maximum. On peut imaginer d'utiliser en plus l'attribut "lang" pour encore mieux cibler les lecteurs potentiels :

```
<-- For speakers of US English -->
<META name="keywords" lang="en-us" content="vacation, Greece, sunshine">
<-- For speakers of British English -->
<META name="keywords" lang="en" content="holiday, Greece, sunshine">
<-- Pour lecteurs francophones -->
<META name="keywords" lang="fr" content="vacances, Gr&egrave;ce, soleil">
```

Tous ces tags META peuvent être définis à la dure dans le code HTML. Mais on peut aussi utiliser pour certains

Format → Page Title and Properties

qui fait apparaître la boîte de dialogue à trois zones d'entrées permettant de définir respectivement le titre, le nom de l'auteur (meta "author") et une description (meta "description") :



Après fermeture de la boîte, on peut constater que les tags sont bien présents dans le code HTML :

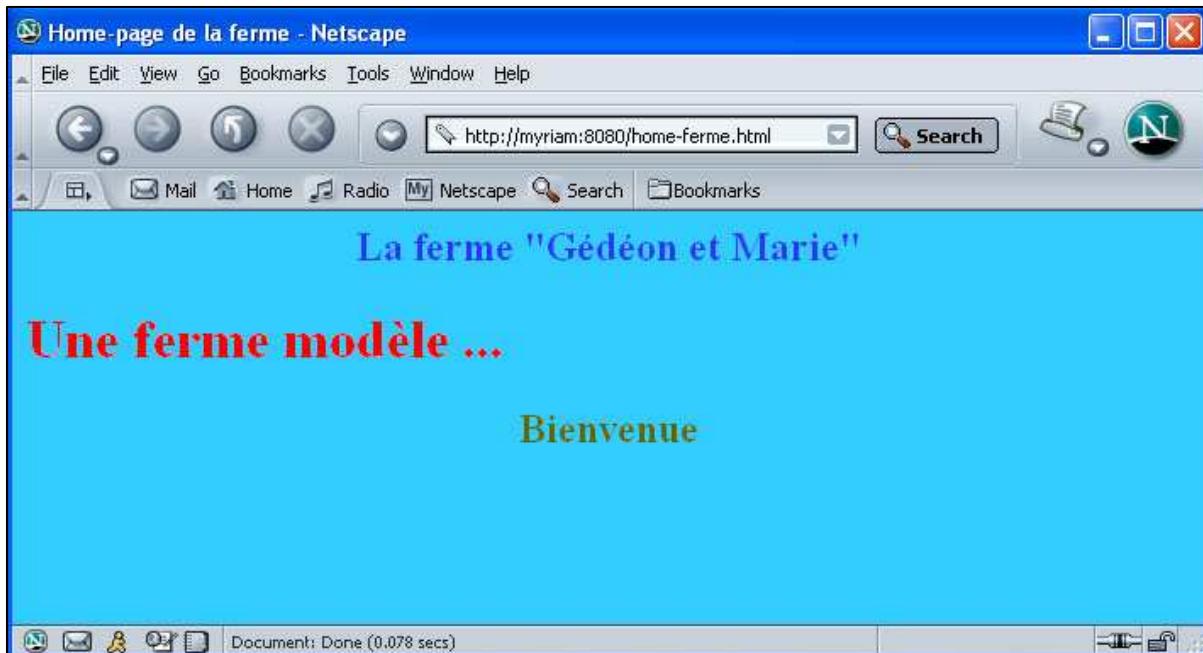
home-ferme.html (2)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Home-page de la ferme</title>

<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
<meta name="author" content="G&acute;d&acute;on Lavinasse">
<meta name="description" content="Ferme, Cultures, Elevage">
</head>

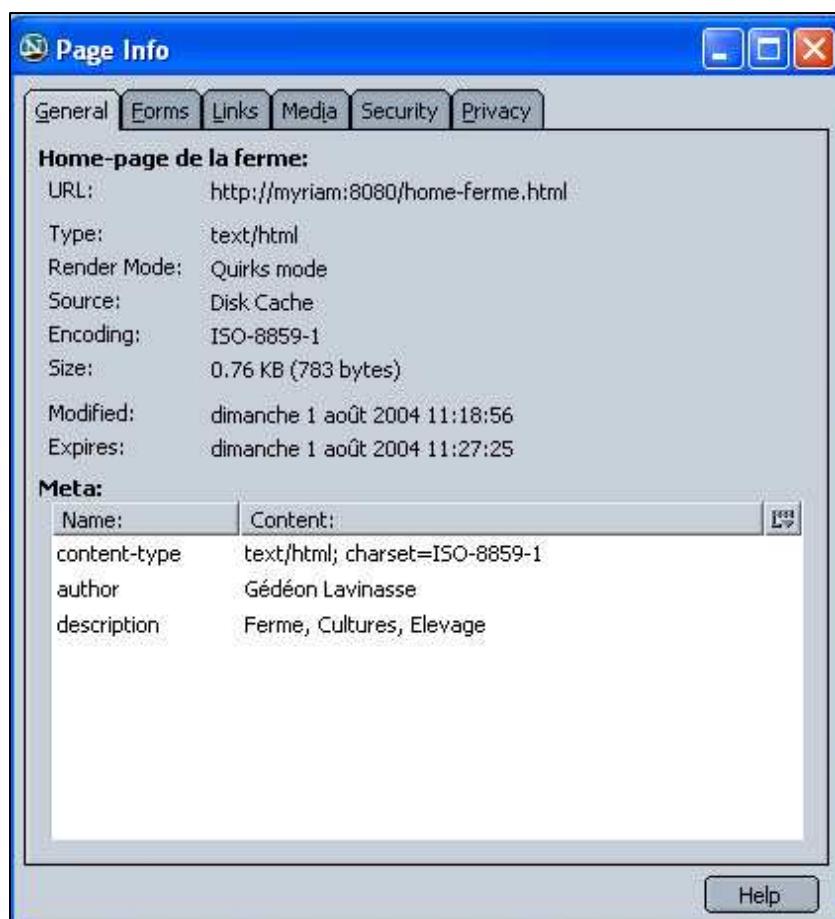
<body text="#000000" bgcolor="#33ccff" link="#000099" vlink="#990099">
...
</body>
</html>
```

La home page de notre site peut à présent être installée sur le serveur. En pratique, cela se fait en utilisant le protocole FTP avec un logiciel quelconque (WS_FTP Pro, Reflection, etc). Dans notre cas, avec Tomcat en attente sur le port 8080 sur une machine myriam :



Les meta-information peuvent être obtenues par

View → Page info



10.Au-delà de HTML

10.1 SGML

En fait, le langage HTML est un sous-ensemble d'un langage plus vaste appelé **SGML** (Standard Generalized Markup Language). Celui-ci est un standard international permettant de définir la représentation des textes électroniques indépendamment des machines et des systèmes d'exploitation sous-jacents.

En fait, SGML est un métalangage, en ce sens qu'il peut décrire des langages – en fait, un langage à *balises* ou *marqueurs*, ces dernières permettant de spécifier comment le texte doit être manipulé. Un tel langage à balises n'est pas un concept neuf : les typographes en connaissent depuis longtemps. SGML permet donc de définir les balises de tels langages, les enchaînements de balises obligatoires, la manière de reconnaître ces balises.

Plus précisément, un document SGML comporte notamment une partie déclaration (elle spécifie le jeu de caractères utilisés, les caractères qui ont une signification particulière) et, bien sûr, le document proprement dit, avec ses balises

10.2 XML

Le langage **XML** (Extensible Markup Language) est un sous-ensemble de SGML. Tout comme lui, il est extensible et non pas figé comme HTML. Le principe qu'il applique est toujours qu'il décrit un document au moyen de balises. Plus précisément, un document XML comporte une partie déclarative de type SGML, la définition du type de document et le document lui-même, décrit au moyen des balises. Un exemple de document XML est :

listeSociétés.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!—Created by Vilvens on 19 avril 2002, 17:00 -->
<!DOCTYPE listeSociétés [
    <!ELEMENT listeSociétés ANY>
    <!ELEMENT société (nomSociété, adresse)>
    <!ELEMENT nomSociété (#PCDATA)>
    <!ELEMENT adresse (rue_numéro, codePostal_ville)>
    <!ELEMENT rue_numéro (#PCDATA)>
    <!ELEMENT codePostal_ville (#PCDATA)>
]>
<listeSociétés>
    <société>
        <nomSociété>Genius S.A.</nomSociété>
        <adresse>
            <rue_numéro>rue de la Réussite 69</rue_numéro>
            <codePostal_ville>4500 Visé</codePostal_ville>
        </adresse>
    </société>
</listeSociétés>
```

XML possède de multiples possibilités⁸ – mais ceci nous écarte de notre sujet ...

⁸ voir par exemple "Langage Java (III) : Programmation des applications Web" – du même auteur

10.3 XHTML

A priori, HTML est un langage assez laxiste. Autrement dit, une page HTML rédigée en ne respectant pas entièrement les règles peut être interprétée correctement par bon nombre de browsers. Cependant, la multiplication des supports utilisés pour surfer sur Internet (ordinateurs divers, mobiles en tous genres comme les pockets PC et les GSM) a amené le besoin de pages confectionnées avec plus de rigueur, car bon nombre de ces supports ne disposent pas des ressources suffisantes pour interpréter une page HTML rédigée de manière imprécise. En fait, HTML définit comme afficher des données. Comme XML permet de décrire ces données avec précision, il a semblé assez logique de marier les deux langages pour produire XHTML (EXtensible HyperText Markup Language). Ainsi, en attendant des browsers capables de lire du XML pur, XHTML fournit des pages décrites de manière rigoureuse selon un formalisme XML, pages qui resteront donc toujours compatibles et utilisables pour tous les browsers. XHTML est donc destiné à remplacer le HTML classique, en fournissant une version plus stricte et plus clairement définie, par ailleurs assez semblable à HTML 4.01. Autrement dit, il s'agit de HTML défini comme une application de XML. XHTML 1.0 a fait l'objet d'une recommandation du W3C dès 2000. La DTD du XHTML est spécifiée en SGML.

Quelles sont les principales différences entre XHTML et HTML ? Tout d'abord, tous les tags et tous les attributs doivent être écrits en caractères minuscules et les valeurs des attributs doivent toujours être placées entre guillemets. De plus,

- ◆ les tags doivent être emboîtés correctement : donc la séquence `<i>` doit être refermée par `</i>` et pas `</i>`;
- ◆ tout tag non vide doit être refermé : donc `<p>` doit être accompagné d'un `</p>`
- ◆ tout tag vide doit être fermé : donc `<hr>` doit devenir `<hr/>` (et même `<hr />` : l'espace supplémentaire assure la compatibilité avec les browsers les plus récents);
- ◆ l'attribut name est remplacé par l'attribut id; on écrit donc
``
mais on peut garder la compatibilité avec les browsers anciens en écrivant :
``

La structure de base d'un document XHTML précise que tout document XHTML doit posséder au minimum les éléments suivants :

squelette.htm

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Document sans titre</title>
</head>
<body>
</body>
</html>
```



Reste encore HTML 5 – mais nous y reviendrons. D'abord, pourquoi parle-t-on de "toile d'araignée mondiale" (World Wide Web) ? A cause des liens hypertextes ...

III. Les liens

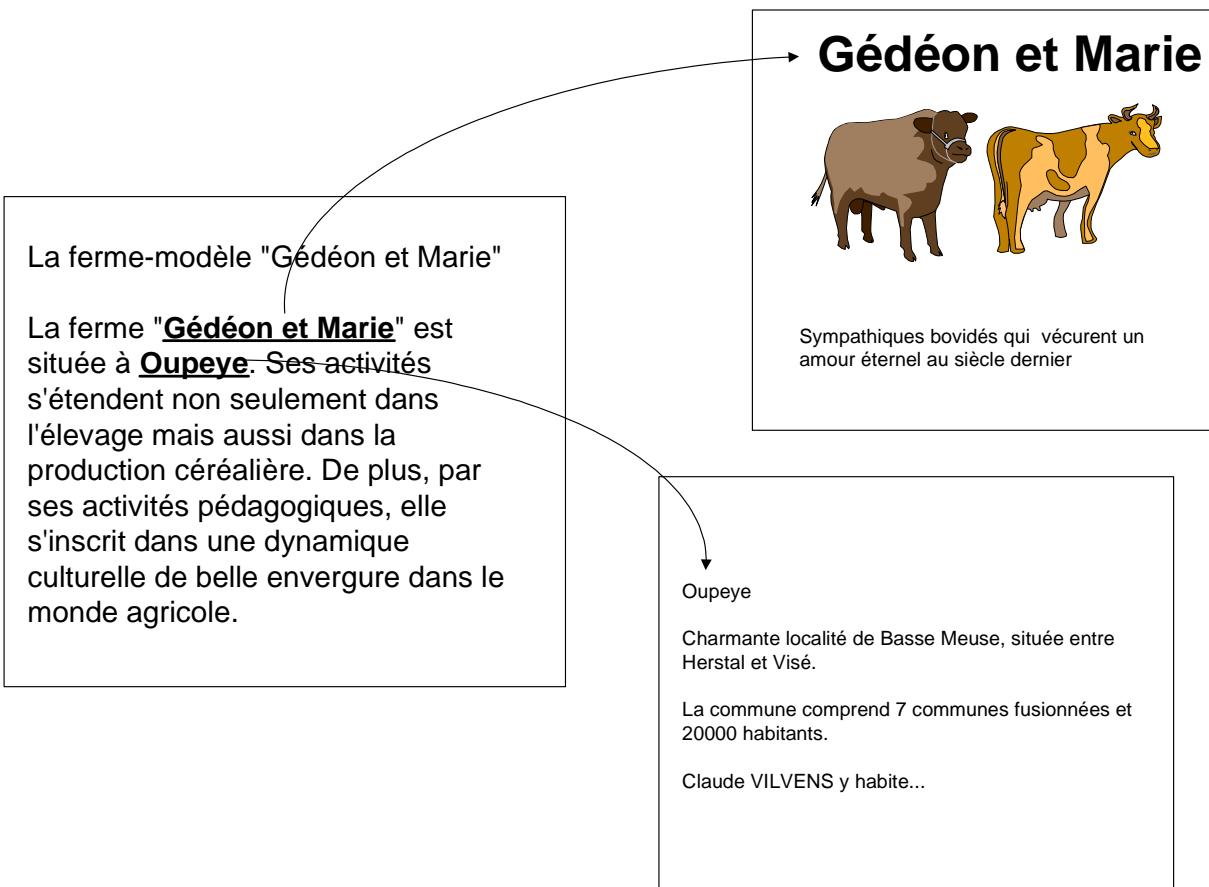


Tout homme qui a été professeur garde en lui quelque chose de l'écolier.

(A. de Vigny, Mémoires inédits)

1. L'hypertexte de base

Le passage d'une page à une autre par un simple clic sur un mot ou groupe de mots est l'une des raisons du succès d'Internet. Pour rappel, le principe est le suivant. Un document hypertexte se présente à priori comme un document normal : titres, paragraphes, listes à puces, tableaux, images, ... Ce qui le particularise, ce sont les **liens**. Un lien a, le plus souvent, l'apparence d'un mot (ou groupe de mots) souligné (en mode graphique) ou en vidéo inverse (mode texte); on peut généraliser en créant un lien à partir d'une image. Un clic de la souris (mode graphique) ou un Enter (mode texte) sur le mot fait passer automatiquement à un autre document ou à un autre emplacement du même document. Celui-ci, normalement, apporte des informations complémentaires sur le mot ou présente la suite d'une présentation ou encore permet de revenir au début du sujet. Par exemple, pour un site agricole :



En fait, selon les volontés du concepteur, il est possible de se déplacer n'importe où. Ce qui n'est pas toujours conseillé, l'utilisateur pouvant finir par se perdre dans les méandres des liens existant ... Il est bien sûr toujours possible de se connecter à un autre serveur. Et tout recommence. Ce n'est pas un hasard si naviguer au sein de ces pages en suivant les liens se dit aussi "surfer" ...

Cette fonction d'hypertexte s'implémente facilement en HTML.

Retenant notre exemple de ferme modèle et supposons vouloir apporter une page de justification sur le nom de la ferme, soit "Gédéon et Marie".

On construira tout d'abord la page en question. Mettons qu'elle aura l'aspect suivant :



avec un code HTML qui est le suivant :

home-ferme-etymol.html

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Mozilla/4.7 [fr] (Win98; I) [Netscape]">
<meta name="author" content="C.Vilvens">
<meta name="description" content="Ethymologie">
<title>Pourquoi "G&acute;d&acute;on et Marie" ?</title>
</head>

<body background="lightgreen" bgcolor="#ffffcc" text="#000000"
link="#0000ee" alink="#0000ee" vlink="#551a8b">

<b><i><font color="#0000ff"><font size="+2">La ferme "G&acute;d&acute;on et Marie"</font></font></i></b> <br>

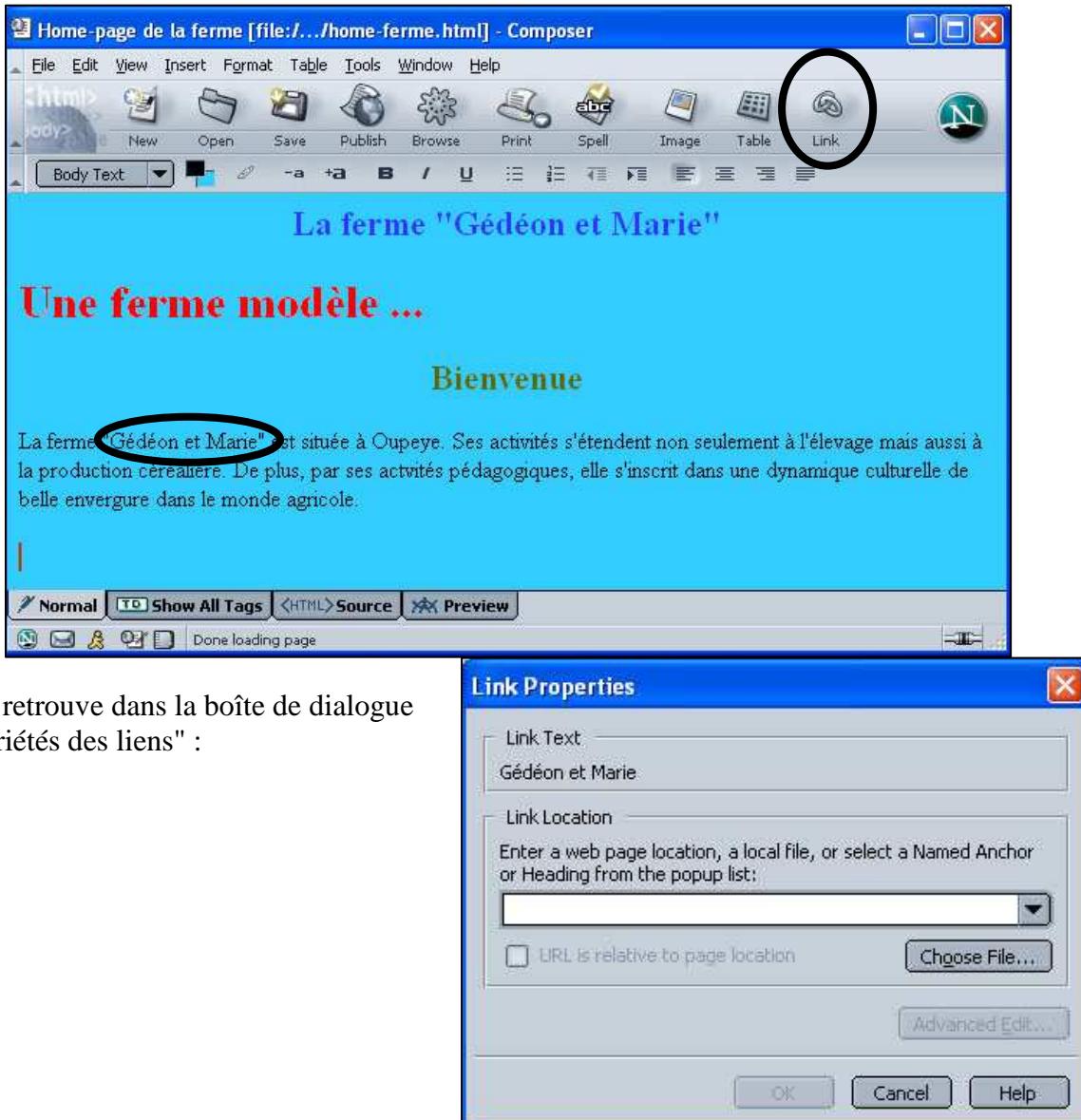
<b><i><font size="+3">Pourquoi ce nom ?</font></i></b>
```

```
<p>Gédon et Marie sont deux sympathiques bovidés qui  
vont curer un amour immortel au siège du dernier .... </p>  
<p><br>
```

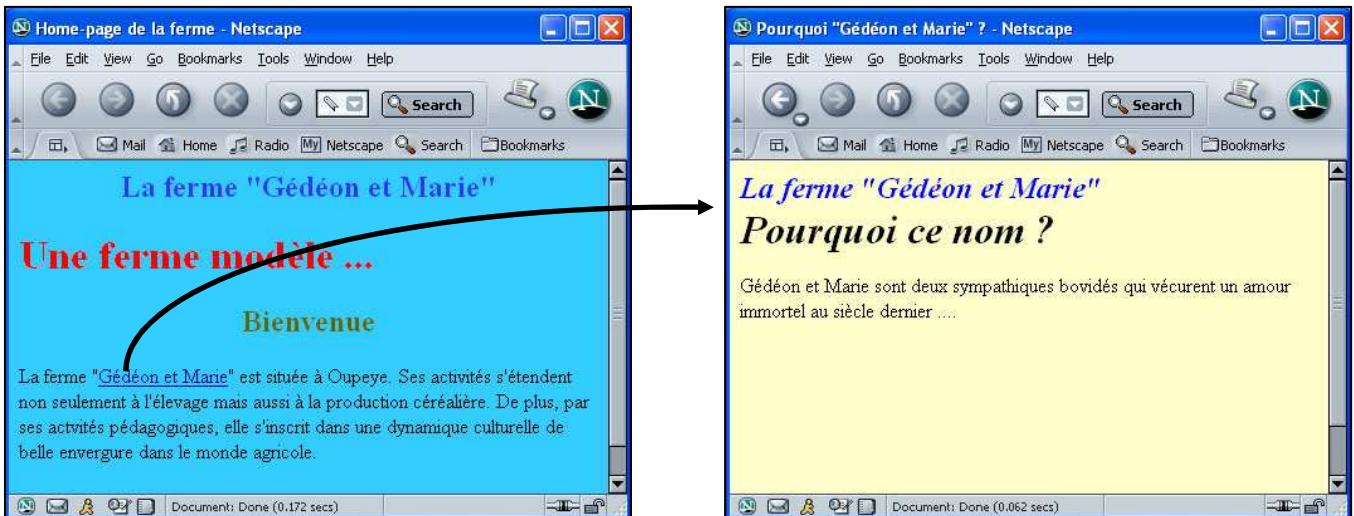
```
</body>  
</html>
```

Supposons qu'il soit sauvé dans un fichier appelé home-ferme-etymol.html.

Reste à établir à présent la relation vers ce fichier à partir de la page de départ. Editons cette page. Il suffit en fait de sélectionner le mot ou groupe de mots chargé de représenter le lien et de cliquer l'icône de lien :



Il suffit dès lors de sélectionner le fichier html qui doit être atteint à partir du lien, soit ici home-ferme-etymol.htm. On obtient ainsi le groupe 'Gédéon et Marie' souligné (si du moins on a gardé la configuration par défaut). On peut tester dans le browser que le lien fonctionne bien :



2. La balise de liens

On vient de générer ainsi un lien donc le code HTML est :

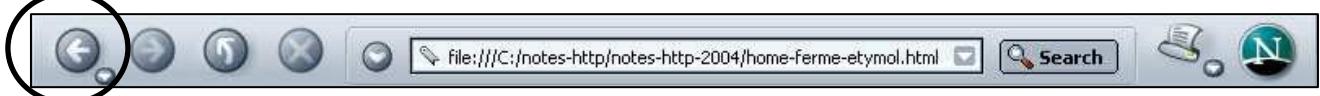
```
<A HREF="home-ferme-etymol.htm">G&acute;d&acute;eon et Marie</A>
```

La balise de base est `<A>`. Elle doit être complétée d'attributs. Le plus important et le plus utile est **HREF** (pour **Hypertexte REference**); il doit être suivi d'un = et du nom du fichier contenant la page sur laquelle il faut se connecter. Dans notre exemple, on trouve :

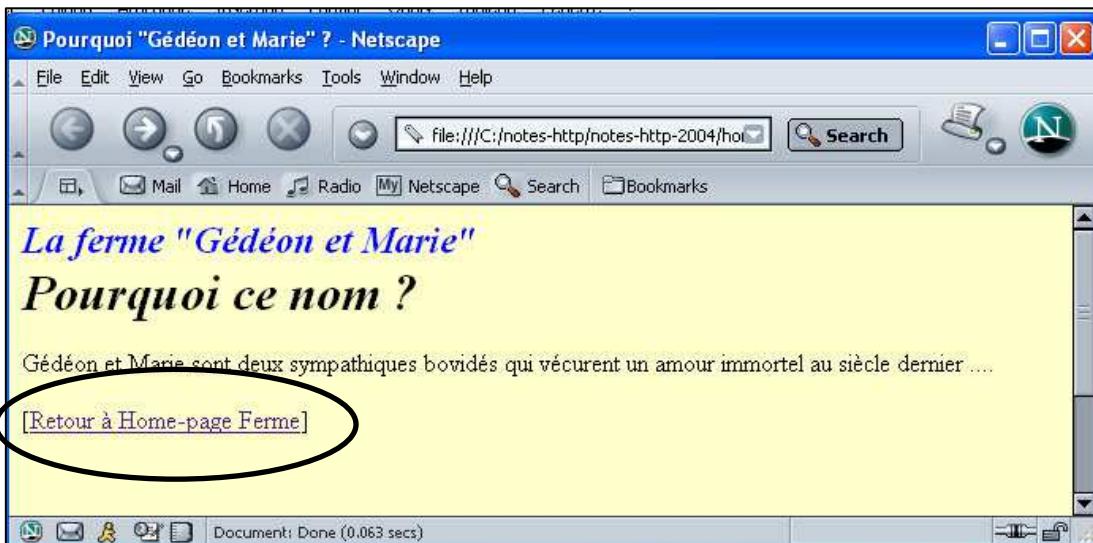
```
<A HREF="home-ferme-etymol.htm">
```

Le texte qui suit la balise n'en fait pas partie : il s'agit en fait du texte qui matérialise le lien en étant souligné. C'est sur ce texte qu'il convient de cliquer pour changer de page.

Bien sûr, une question se pose : comment revenir en arrière ? Dans notre cas, c'est très simple : en cliquant sur le bouton "Précédent" [Back] :



Cependant, ceci peut se révéler insuffisant lorsque l'on a navigué au travers de tout un ensemble de pages : le "Précédent" ramène simplement à la page accédée précédemment, pas forcément à la page d'entrée ou, surtout, à la page de niveau supérieur ! Il sera donc prudent de toujours prévoir un lien permettant le retour à la page introductory. Dans notre cas, on ajoutera un point de retour comme celui-ci :



dont le code HTML est simplement :

```
<P>[<A HREF="home-ferme.htm">Retour &agrave; Home-page Ferme</A>]</P>
```

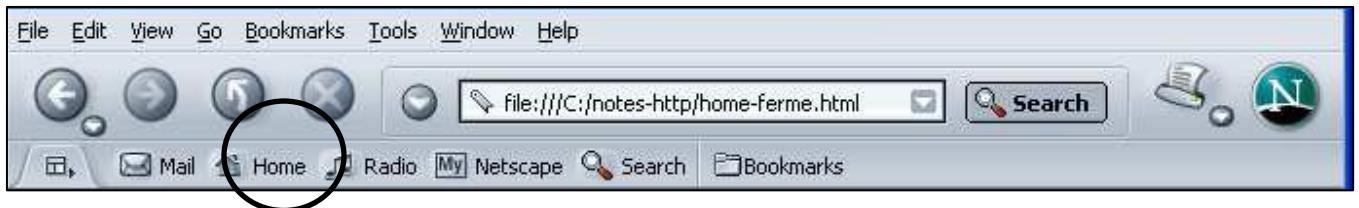
Remarques

1) Il convient de s'assurer que les balises ne s'entrecroisent pas au sein d'une ligne. Ainsi,

```
<P>[<A HREF="home-ferme.htm">Retour &agrave; Home-page Ferme</P>]</A>
```

n'est pas correct.

2) On peut retourner, en cours de navigation, à la home-page de connexion en cliquant sur le lien "Accueil" [Home] :



3. Les chemins et les URLs

Dans l'exemple précédent, le nom du fichier visé par le lien est dépourvu de toute indication de chemin. Implicitement, il est supposé se trouver dans le répertoire de la home-page. Il pourrait ne pas en être ainsi, auquel cas la précision du chemin est nécessaire.

Imaginons ainsi que la page pointée ne se trouve pas dans le répertoire courant, mais dans le sous-répertoire appelé "details". En utilisant l'outil de création de liens, on obtiendra un code HTML :

```
<A HREF="details/home-ferme-etymol.htm">G&eacute;d&eacute;on et Marie</A>
```

A remarquer que le chemin (relatif au répertoire courant) utilise la notation UNIX pour les chemins (soit un / et pas un \). Un passage par le browser indique que le passage à la deuxième page fonctionne - mais il n'en est pas de même du retour à la home-page par le lien

que nous avons créé précédemment ! En fait, le navigateur a changé de répertoire et y cherche la home-page, bien sûr sans la trouver ... Il faut donc rectifier le lien pour que son code HTML soit :

```
<P>[<A HREF="../home-ferme.htm">Retour &agrave; Home-page Ferme</A>]</P>
```

On pourrait préciser l'URL complète :

```
<A HREF=" file:///C:/htmep1.net/courshtm/home-ferme.htm">
```

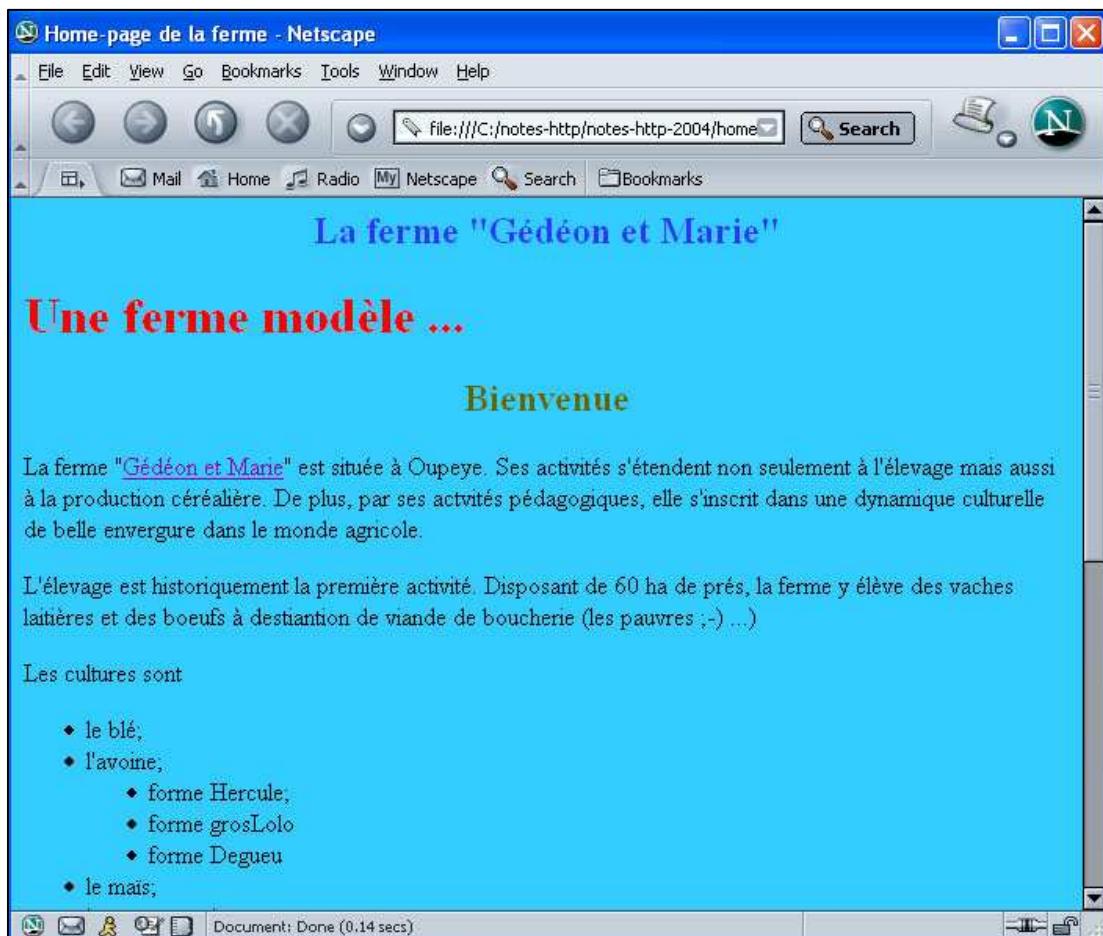
De cette manière, on peut ainsi créer des liens vers d'autres sites WEB :

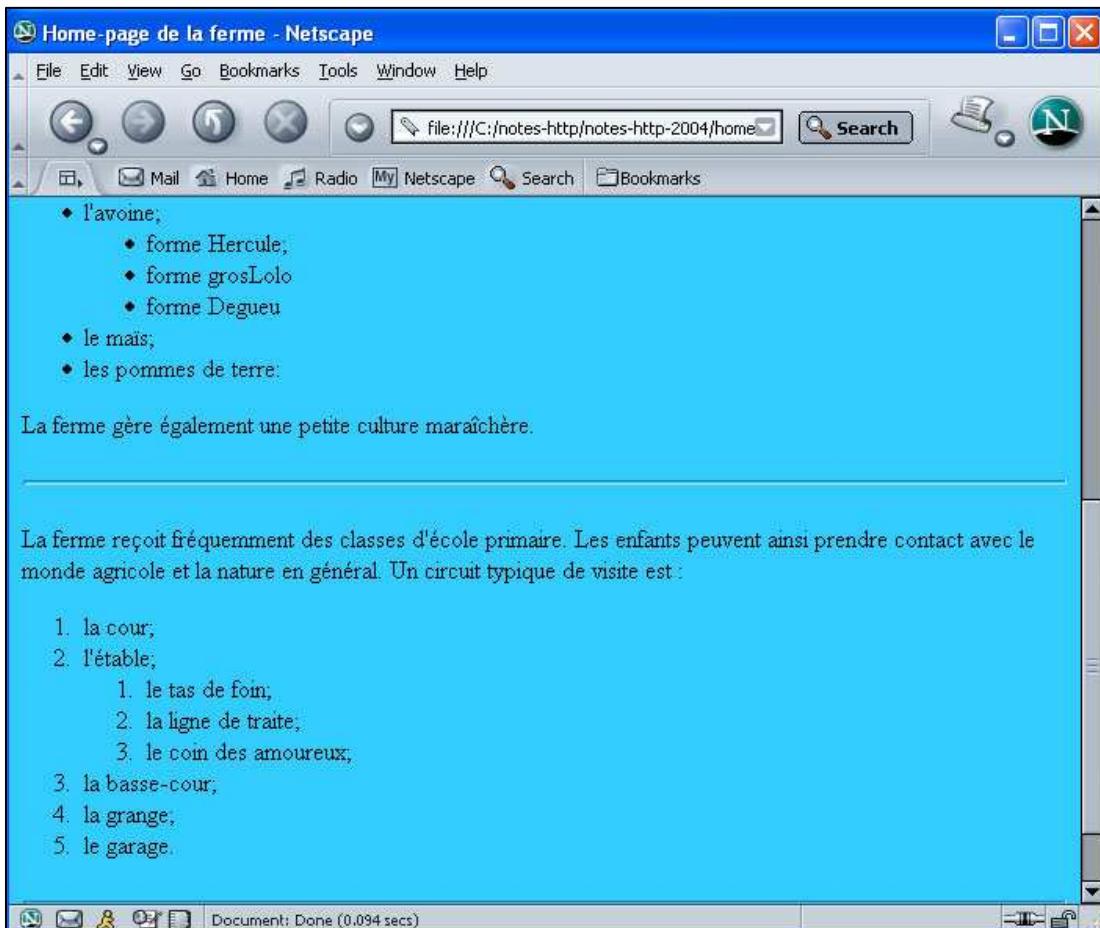
```
<A HREF="http://www.belnet.be">
```

4. Les signets

On aboutit dans une nouvelle page à son début. Mais, parfois, on peut souhaiter revenir dans une autre page à une position donnée. De même, on peut souhaiter voyager au sein d'une même page, par exemple parce qu'elle est très longue. On utilise dans ce cas des *signets* (*anchors* en anglais).

Supposons ainsi que notre home page se soit allongée (ce qui n'est pas forcément une bonne idée) :



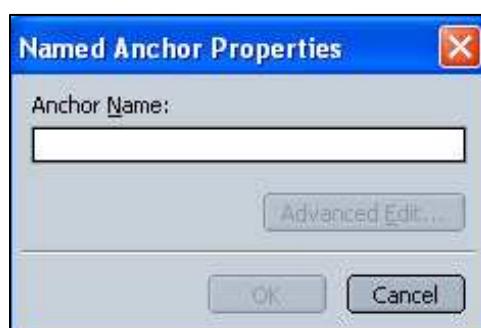


Il peut être utile de prévoir un lien qui permet de passer directement sur les activités pédagogiques de la ferme. Pour ce faire, il faut :

- ◆ définir un signet à l'endroit souhaité : après avoir positionné le curseur à cet endroit, il suffit de sélectionner dans le menu :

Insert → Named anchor

L'éditeur demande alors un nom pour le signet :



Appelons-le, par exemple, 'école'. Une petite icône s'affiche à l'emplacement du signet :

 La ferme reçoit fréquemment des classes d'école primaire. Les enfants peuvent ainsi prendre contact avec le monde agricole et la nature en général. Un circuit typique de visite est :

Le code HTML généré utilise la balise <A> avec l'attribut NAME= suivi du nom du signet :

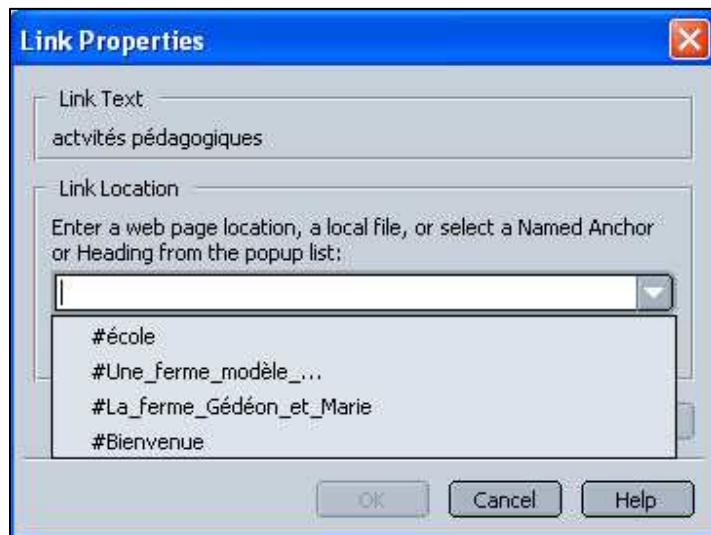
ou plutôt

< A NAME ="école">

- ♦ créer le lien vers le signet. Ici, on procède de manière similaire à un lien vers une autre page. Après avoir sélectionné le groupe de mots siège de l'hyperlien (ici, "activités pédagogiques") :

La ferme "[Gédéon et Marie](#)" est située à Oupeye. Ses activités s'étendent non seulement à l'élevage mais aussi à la production céréalière. De plus, par ses **activités pédagogiques**, elle s'inscrit dans une dynamique culturelle de belle envergure dans le monde agricole.

on clique sur le bouton de liens pour obtenir :



Mais au lieu de choisir un fichier, on choisit un nom de signet parmi ceux proposés dans le menu déroulant de la boîte de dialogue. Le lien est ainsi établi de la même manière qu'un lien vers une page. Le code HTML généré est :

De plus, par ses activités pédagogiques, elle s'inscrit dans ...

On retrouve le tag <A> et l'attribut HREF; simplement, la distinction entre un nom de fichier et un signet est marquée par la présence d'un préfixe '# '.

Il est possible de combiner les deux en fournissant une référence vers un signet dans un autre fichier. Ainsi, pour que le lien de retour se fasse au début de la home-page, le code HTML généré sera :

<P>[Retour ` Home-page Ferme]</P>

Remarque

Il est aussi possible de créer des liens à partir d'images. Nous en reparlerons dans le chapitre suivant.



Que serait une page HTML sans images ? C'est, du moins, ce que se disent les Internautes utilisant des PCs Windows, des Macs ou des stations Solaris. Donc, voyons comment inclure des telles images.

IV. Les images



La femme sera toujours le danger de tous les paradis.

(P. Claudel, Conversations dans le Loir-et-Cher)

1. Les images utilisées sur le WEB

Il existe en informatique de nombreux types d'image différents. On parle encore de formats d'images. Rares sont les logiciels qui les manipulent tous, malheureusement.

Les fichiers images les plus courants sont les fichiers 'bitmap', dont l'extension la plus connue est **.bmp**. Ils contiennent tout simplement le codage de tous les pixels de l'image, c'est-à-dire la position et la couleur de chacun d'entre eux. Le nombre de bits nécessaires varie selon le nombre de couleurs disponibles. Il s'agit donc en quelque sorte d'une photographie numérique de l'image. Tout redimensionnement de l'image affectera la qualité de celle-ci. Parmi les autres formats d'images bitmaps, citons encore **gif** de Compuserve, **jpg** de Join Photo Expert Group, **mac** de MacPaint, **pcd** de Kodak, **psd** de Photoshop et **tiff** d'Aldus Corporation.

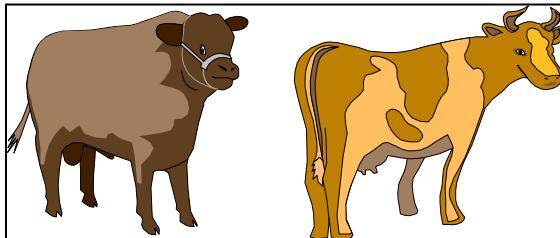
Le format **GIF** est le format que tout interface de navigation graphique doit reconnaître. Les plus répandus parmi ces navigateurs graphiques sont également capables d'afficher des images au format **JP(E)G**.

Une image vectorisée est mémorisée sous forme d'un ensemble de formes géométriques dont la combinaison forme l'image. Le format **cdr** de CorelDRAW! en est un exemple. Cette façon de procéder présente deux avantages :

- ♦ la taille des fichiers correspondants est plus réduite que celle des fichiers bitmaps;
- ♦ la modification de la taille des images se fait sans perte de qualité.

2. L'insertion d'une image

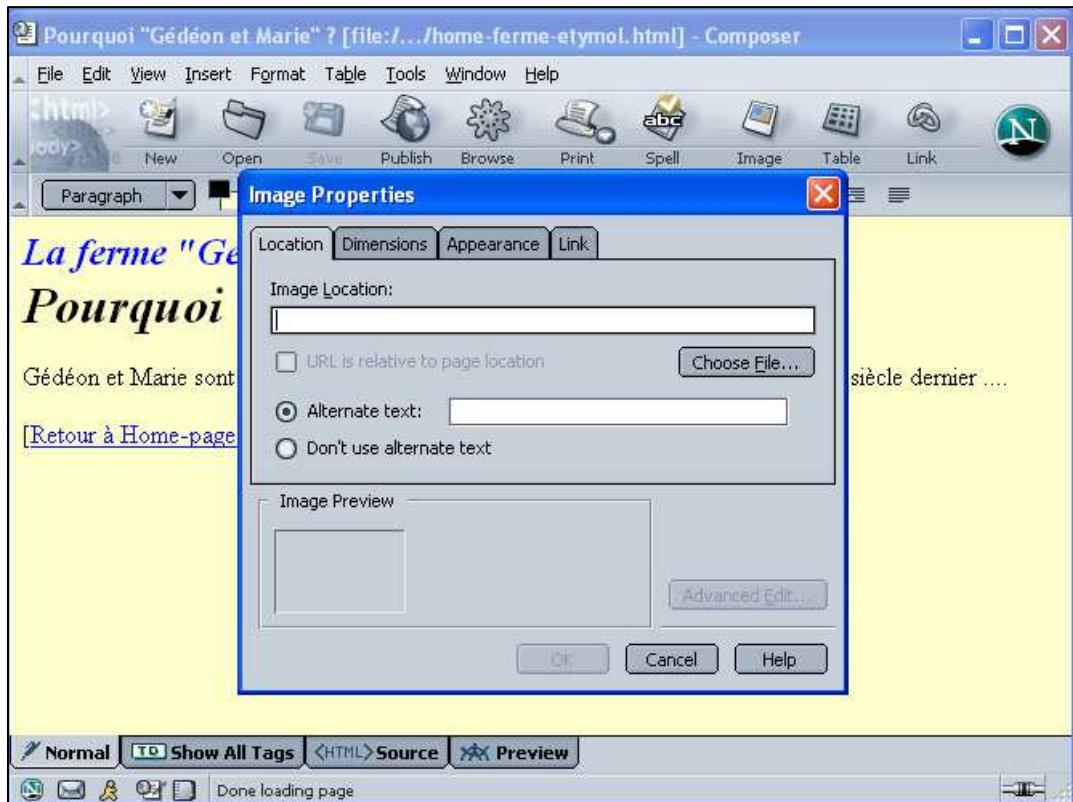
Reprenant notre exemple de ferme modèle, supposons donc vouloir enrichir la page apportant la justification du nom de la ferme, soit "Gédéon et Marie". Une image valable serait celle-ci (son origine est un logiciel quelconque, pourvu qu'il manipule les images) :



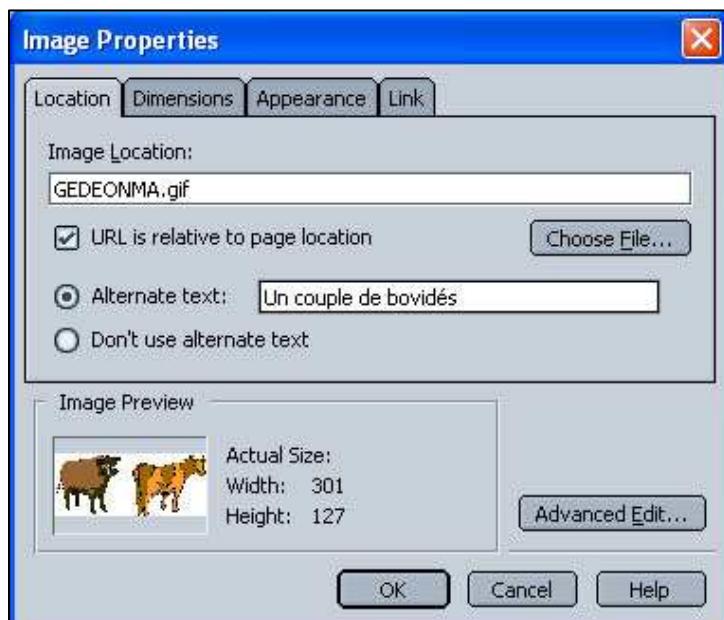
Supposons qu'elle soit contenue dans le fichier **gedeonma.gif**. Pour l'insérer au sein de la page considérée, il faut d'abord la charger dans l'éditeur. Après s'être positionné à l'endroit où l'on souhaite l'image, on clic sur l'icône d'image :



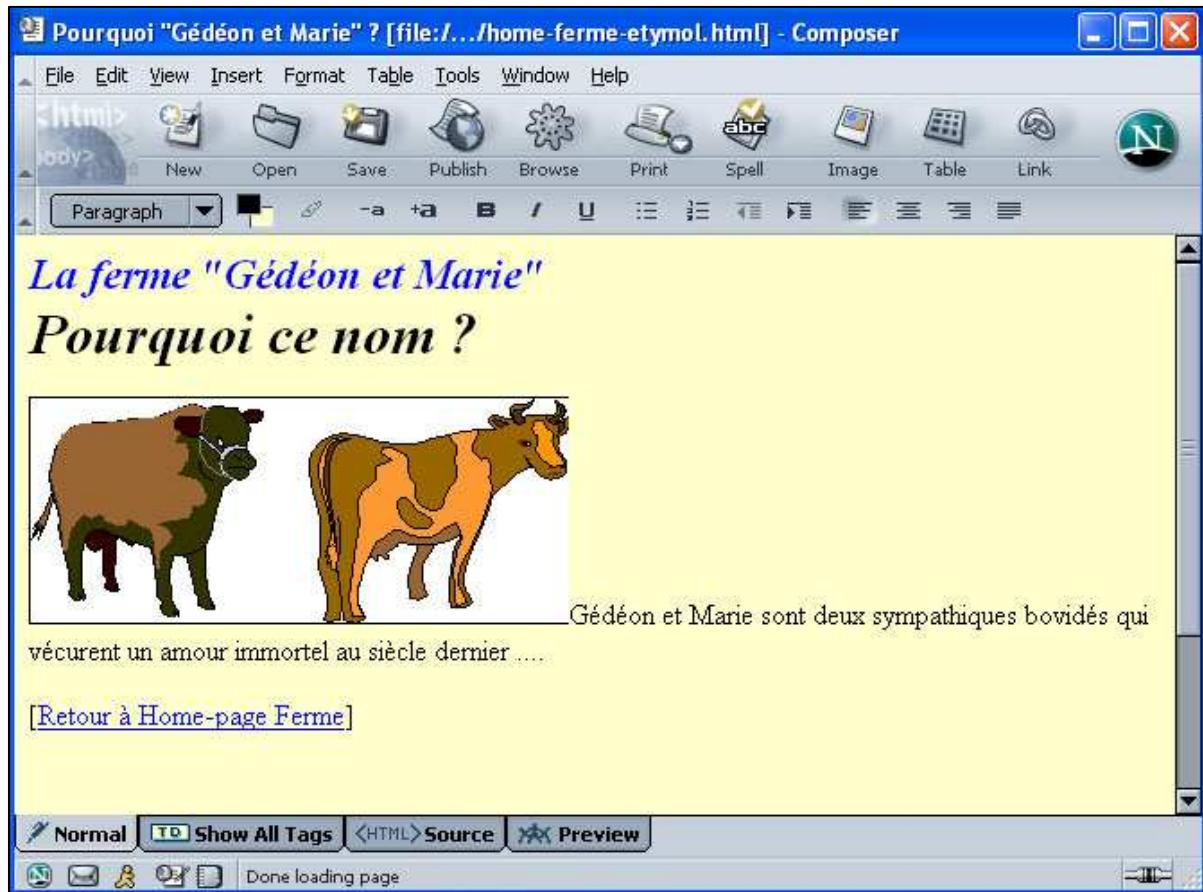
On obtient une boîte de dialogue permettant de caractériser l'image désirée :



En choisissant ensuite les boutons Ouvrir puis OK, et après avoir fourni un texte destiné à être affiché à la place de l'image pour les browsers qui ne sont pas capables de l'afficher,



on obtient :



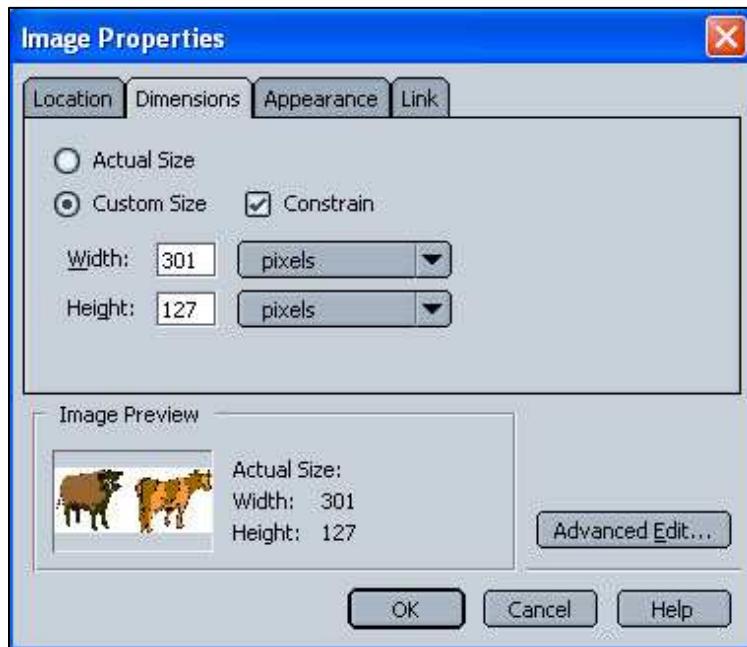
| <P></P>

ce qui indique que l'insertion d'une image est codée par le tag **** et que celui-ci possède divers attributs.

Evidemment, on peut trouver que l'image est un peu grande ou au contraire trop petite. Pour en rectifier la taille, par exemple, il suffit de sélectionner l'image et de choisir "Propriétés de l'image" [Image properties] dans le menu obtenu par un clic droit sur l'image :



La boîte de dialogue obtenue est celle déjà rencontrée :



et permet de

- ◆ fixer de nouvelles dimensions;
- ◆ choisir l'épaisseur du cadre;
- ◆ fixer la position d'un texte d'accompagnement éventuel.

Une modification et un bord léger donnent le code :

```
<IMG SRC="GEDEONMA.gif" alt="Un couple de bovidès" WIDTH="301"  
HEIGHT="127" BORDER="1">
```

A remarquer que l'on peut aussi définir une image à faible résolution (qui peut donc être chargée rapidement) qui sera affichée en attendant que l'image principale (probablement plus lourde) soit chargée. A ce propos ...

3. La taille mémoire des images⁹

3.1 Position du problème

L'un des problèmes habituels que l'on rencontre lorsque l'on utilise des images sur Internet est leur taille, non pas la taille longueur*largeur, mais leur taille en termes de bytes occupés (on parle parfois de leur **poids**, par opposition à la taille géométrique). Si une image représente 100 ou 200 Ko, le temps nécessaire pour la charger depuis le serveur sur une machine client sera trop long pour être supportable pour l'utilisateur ... qui ira voir ailleurs ! Très souvent, ces images sont le résultat d'un scan, qui s'est voulu de haute définition, et qui fournit une image de l'ordre de plusieurs centaines de Kilobytes (pour ne pas dire plus). Même en passant du format classique .bmp au format .jpg ou .gif, le résultat reste imposant. Il importe donc, à partir de cette image, certes de très bonne résolution mais trop grosse, de créer **une image plus légère, peut-être moins parfaite, mais suffisante dans le contexte Internet.**

⁹ Ce paragraphe est grandement redéivable aux communications de l'un de nos Anciens diplômés, Philippe Lesire

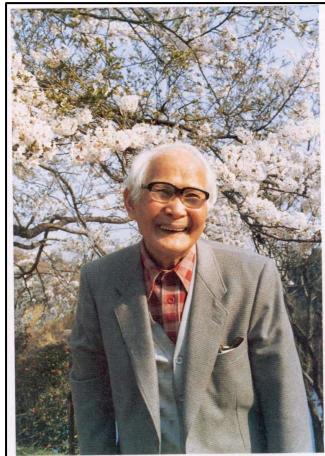
Sur quels facteurs peut-on intervenir ? Bien logiquement, les éléments suivants interviennent :

- ◆ la taille à l'affichage;
- ◆ le nombre de couleurs;
- ◆ le type de format.

C'est donc sur eux qu'il faudra agir, au moyen d'un logiciel de manipulation d'images. En ce qui nous concerne, nous utiliserons **Paint Shop Pro** (la version 3 est antique, mais est déjà suffisante – les versions suivantes sont évidemment plus riches – et puis il y a **Photoshop** ;-)).

3.2 La taille

On commence par donner à l'image la taille qu'elle doit avoir réellement sur la page HTML. Le principe est de rendre l'image floue avant de la réduire; c'est logique, cet aspect flou disparaîtra lorsque l'image sera rapetissée. Considérons l'image suivante, représentant un sympathique collaborateur Japonais sur fond de cerisiers en fleurs :



Elle "pèse" 796 Ko ! La marche à suivre est la suivante :

- ◆ s'assurer que l'image est bien en 16 millions de couleurs (au besoin, incrémenter ce nombre par Colors->Increase color depth); de cette manière, les algorithmes ne se contenteront pas de supprimer un pixel sur 100;
- ◆ rendre l'image floue (Image->Normal filters); le facteur de filtrage est de l'ordre suivant :

facteur de réduction	filtre
1/2	Soften
2/3	Soften more
plus de 2/3	Blur ou Blur more

Le résultat est une image de 483 Ko – c'est mieux, mais pas assez !

- ◆ effectivement modifier la taille (Image->Resample) pour lui donner celle qu'elle devra avoir sur la page HTML. Pour notre exemple, cela donne une image de 60 Ko.
Si le résultat est trop flou, on peut toujours réaugmenter le contraste
(Image->Normal filters->Sharpen)

3.3 Le format

Habituellement, une photo sera présentée au format .jpeg alors qu'une dessin le sera dans le format .gif. Les images jpeg sont beaucoup moins lourdes que leur correspondant en gif, sauf pour les images de petites dimensions (32x32). Par contre, la netteté est meilleure avec le format .gif, ce qui explique sa préférence pour les schémas.

3.4 Les couleurs

a) Pour les gif

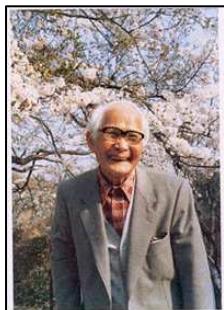
De toute manière, il faut se limiter à 256 couleurs. On utilisera donc la commande Colors->Decrease Color Depth. Deux algorithmes sont possibles :

- ◆ Nearest Color : résultat satisfaisant quand l'image ne comporte pas trop de couleurs;
- ◆ Error Diffusion : plutôt indiquée dans l'autre cas.

b) Pour les jpeg

Une fois revenu à 256 couleurs, il reste à régler le taux de compression, que l'on fixe par le nombre de DPI conservés à l'enregistrement. Ce paramètre se règle dans File->Preferences->General->Saving. En fait, on procède par essais successifs (par exemple, 225, puis 150 DPI) jusqu'à ce que le résultat soit vraiment trop laid !

Pour notre exemple, l'image :



ne pèse plus que 25 Ko !

4. Le texte accompagnant les images

Il est possible de placer du texte à la gauche ou à la droite de l'image insérée. Mais ce texte ne peut être constitué que d'une seule ligne, les autres étant rejetées après l'image.

En pratique, une fois l'image créée, on se positionne à côté de l'image et on encode le texte. Sa position, qui ne convient pas telle quelle en général, peut être rectifiée en sélectionnant dans le menu flottant associé à l'image les caractéristiques voulues :



- ♦ la zone "Alignement" [Align Text to Image] comporte une série de choix permettant de définir la position du texte relativement à l'image;
- ♦ la zone "Espace autour de l'image" [Spacing] permet de fixer, outre l'épaisseur du bord, le nombre de pixels non utilisés autour de l'image.

Pour notre exemple, on obtient ainsi sans problème :

The screenshot shows a web editor window titled 'Pourquoi "Gédéon et Marie" ? [file:///home-ferme-etymol.html] - Composer'. The toolbar includes File, Edit, View, Insert, Format, Table, Tools, Window, Help, and various icons for New, Open, Save, Publish, Browse, Print, Spell, Image, Table, and Link. The main content area displays the text 'La ferme "Gédéon et Marie"' and 'Pourquoi ce nom ?'. Below this is an image of two cows (one brown, one orange) and the caption '(Le boeuf est à gauche !)'. The text 'Gédéon et Marie sont deux sympathiques bovidés qui vécurent un amour immortel au siècle dernier' is also visible. At the bottom, there's a link '[Retour à Home-page Ferme]'. The status bar at the bottom shows 'Normal', 'Show All Tags', 'HTML Source', 'Preview', and 'Done loading page'.

Le code HTML est :

```
<p><IMG SRC="GEDEONMA.gif" ALT="Un couple de bovid&acute;s" width="301" HEIGHT="127" BORDER="1" ALIGN="middle">
<i><b>(Le boeuf est &agrave; gauche !)</b></i><br>
```

où les attributs de la balise IMG correspondent aux caractéristiques fixées ci dessus.

Remarque

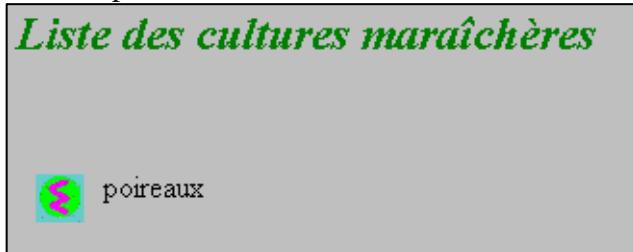
La limite du texte à une seule ligne réduit considérablement les possibilités. Une technique, bien connue des utilisateurs de traitements de textes, consiste à créer un tableau (voir chapitre VI) à bord invisibles.

5. Les images déclencheurs de lien

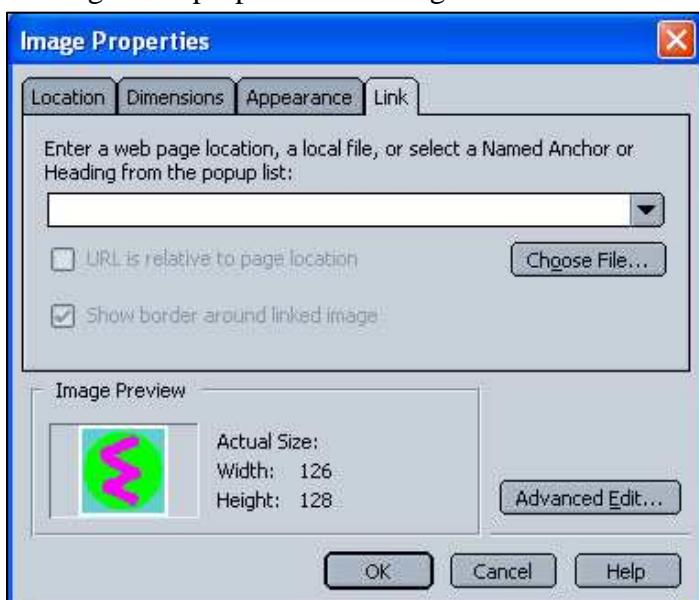
Une image peut servir à déclencher un lien d'hypertexte. Comme il s'agit de l'image dans son entièreté (et pas seulement une zone de celle-ci - voir paragraphe suivant), cette image est en général assez réduite. Prenons par exemple une image de bouton créée au moyen d'un logiciel de dessin :



Prenons une page donnant la liste des cultures maraîchères. Insérons-y le bouton et réduisons sa taille à une dimension acceptable pour un bouton (au moyen des propriétés). Tapons à côté le nom de la première culture et centrons le verticalement. Cela donne :



L'effet souhaité est qu'un clic sur le bouton mène à une autre page (qui, par exemple, donne des détails sur les cultures de la liste). Pour qu'il en soit ainsi, il suffit de sélectionner l'onglet Lien de la boîte de dialogue des propriétés de l'image :



c'est-à-dire quelque chose de fort équivalent à ce que l'on obtient lors de la création d'un lien "normal" - on aurait d'ailleurs pu obtenir le même effet en choisissant le bouton liens dans l'éditeur. Il suffit donc de choisir la page correspondante (et même le signet éventuel). Il suffit d'insérer les autres boutons par copier-coller et de rectifier les liens. On obtient ainsi :

The screenshot shows a web editor window titled "Pourquoi 'Gédéon et Marie'" with the file path "file:///home-ferme-cultures.html". The content is as follows:

La ferme "Gédéon et Marie"

Les cultures maraîchères

-  poireaux
-  navets
-  choux rouges
-  salades

[\[Retour à Home-page ferme\]](#)

At the bottom, there are tabs: Normal, Show All Tags, HTML Source, Preview. The Preview tab is selected.

Two Netscape browser windows are shown side-by-side, both displaying the same content about navets (turnips). Each window has a title bar "Netscape [Home-page ferme]" and a toolbar with standard buttons like Back, Forward, Stop, etc.

The left window displays:

Caractéristiques

Le poireau est du type "long manche", à chair tendre et filandreuse.



Les navets sont destinés aux soupes populaires.



The right window displays:

Caractéristiques

Le poireau est du type "long manche", à chair tendre et filandreuse.



Les navets sont destinés aux soupes populaires.



Le code HTML ainsi généré est :

```
<P><A HREF="details-cultures.htm#poireau"><IMG SRC="bouton.jpg" HSPACE=10
BORDER=0 HEIGHT=25 WIDTH=25 ALIGN=ABSCENTER></A>poireaux</P>

<P><A HREF="details-cultures.htm#navet"><IMG SRC="bouton.jpg" HSPACE=10
BORDER=0 HEIGHT=25 WIDTH=25 ALIGN=ABSCENTER></A>navets</P>

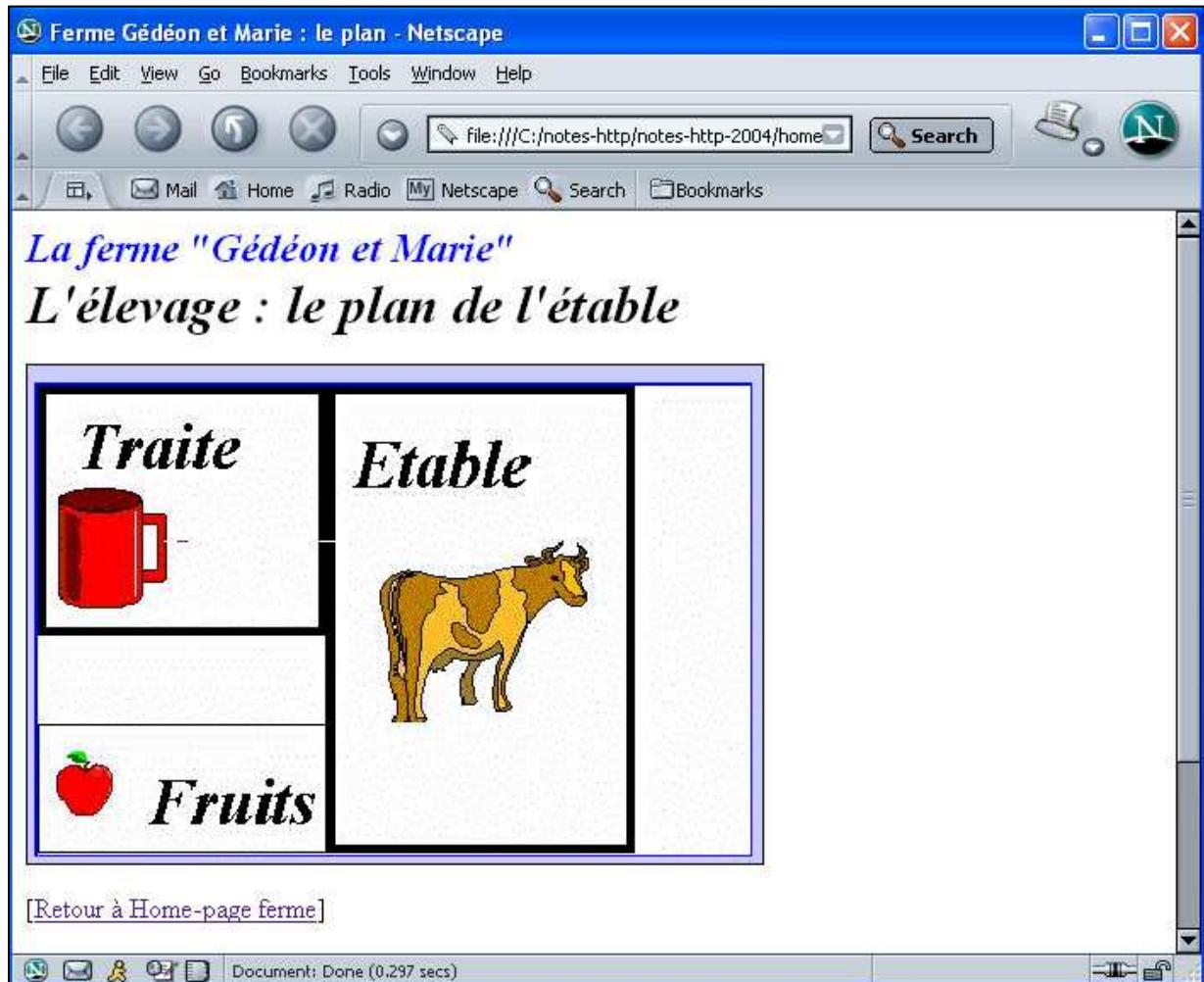
<P><A HREF="details-cultures.htm#choux"><IMG SRC="bouton.jpg" HSPACE=10
BORDER=0 HEIGHT=25 WIDTH=25 ALIGN=ABSCENTER></A>choux
rouges</P>
```

On constate ainsi que le tag de référence <A HREF> est tout simplement complété par le tag .

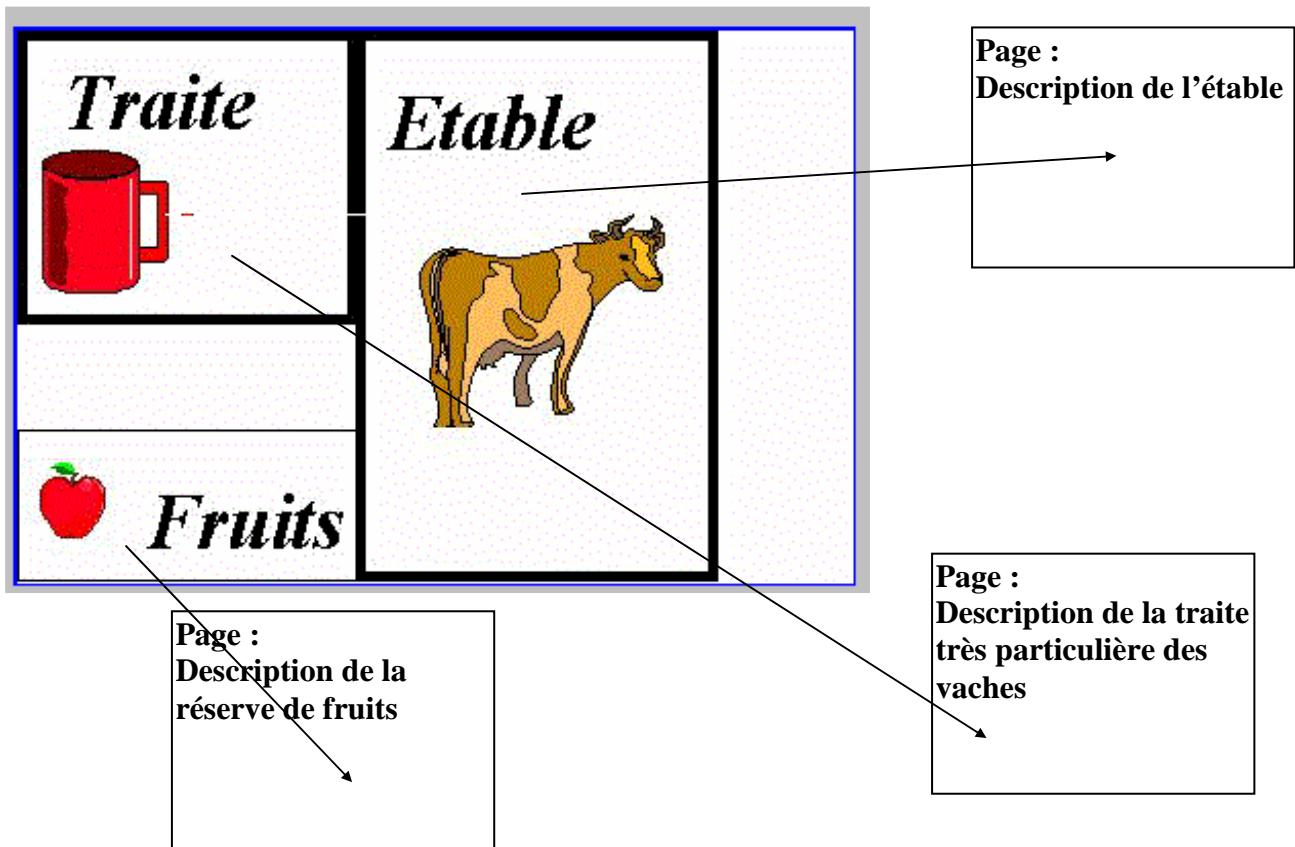
6. Les images référencantes

Une présentation graphique peut aussi être un moyen de proposer des alternatives, tout comme, par exemple, une liste à puces (voir chapitre suivant). La différence réside dans le fait que les choix possibles sont disséminés sur l'image au lieu d'être alignés les uns en dessous des autres.

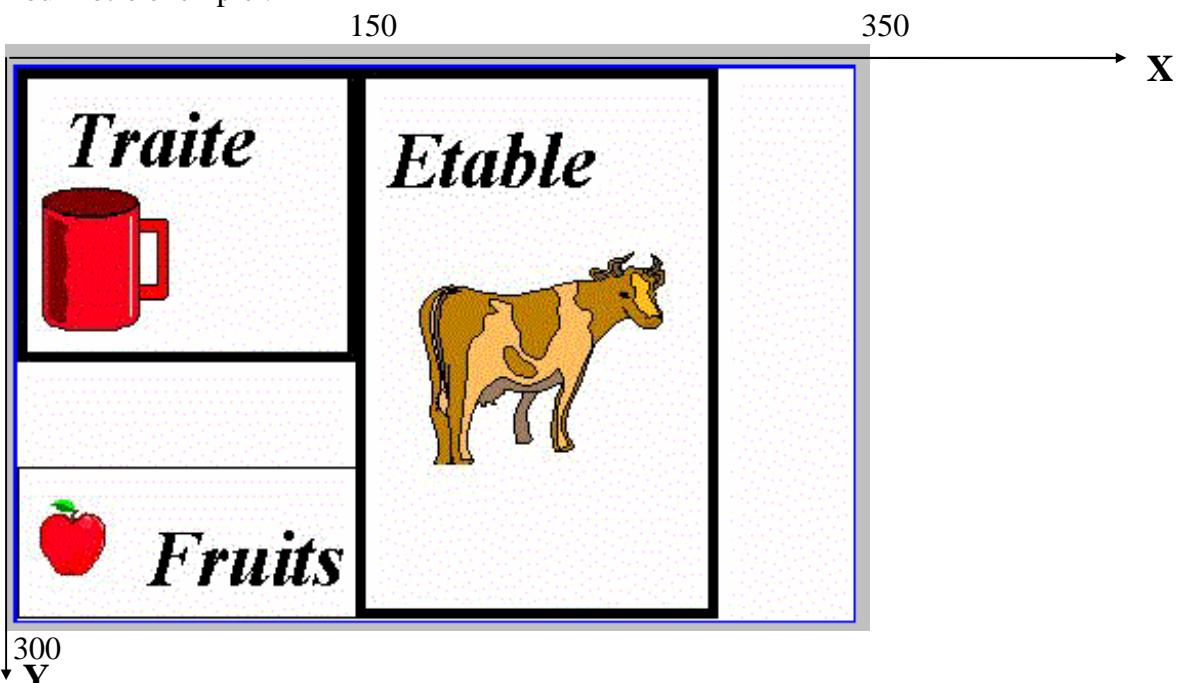
Considérons ainsi la page HTML suivante :



L'effet souhaité est qu'un clic dans la zone de l'étable conduise à une page concernant l'étable, qu'un clic dans la zone de traite mène à une page traitant de la traite, etc



On se doute bien qu'un tel effet ne peut être obtenu qu'en décrivant les différentes zones au moyen des coordonnées de celles-ci. Plus précisément, il faudra fournir les coordonnées des deux points extrêmes de la zone rectangulaire déterminant chaque zone. Le système d'axes est, comme toujours sur les écrans, celui avec un axe Y orienté vers le bas. Pour notre exemple :



Pour obtenir l'effet désiré, il faut :

- ♦ définir un signet spécial, repéré par une balise <MAP>, dont le rôle est de marquer le début d'une zone de description de zones :

```
<MAP name="zones">
```

- ♦ décrire les zones par les coordonnées des points extrêmes au moyen d'une balise <AREA> par zone :

```
<AREA shape="rect" href="vide.htm" coords="10,10,140,140">
<AREA shape="rect" href="vide.htm" coords="180,10,350,300">
<AREA shape="rect" href="vide.htm" coords="10,200,140,300">
```

- ♦ refermer la zone de description :

```
</MAP>
```

- ♦ ajouter au tag de l'image l'attribut **USEMAP** qui indique le signet où l'on trouve la description des zones :

```
<IMG USEMAP="#zones" SRC="plan.jpg" BORDER=2 HEIGHT=269 WIDTH=408>
```

La méthode ainsi décrite est encore appelée **CSIM** (Client Side IMage) : elle n'use que de balises HTML sans appeler de programmes additionnels côté serveur (en fait, des CGIs – voir dernier chapitre).

Bien sûr, en pratique, il est assez fastidieux de devoir déterminer soi-même les coordonnées des zones visées. Il existe fort heureusement des programmes permettant de déterminer de telles zones visuellement. Un bon exemple est **MapThis** (par exemple : <http://www.ccim.be/ccim328/Htmlplus/mapthis.htm>).

7. Trouver ou générer des images

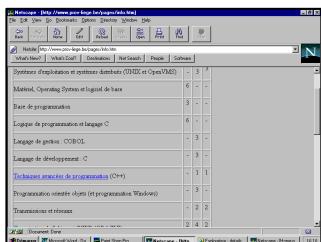
Comment obtenir des images utilisables au sein des documents HTML ? Il est possible d'en obtenir sur les sites Internet, en particulier des gifs animés (en fait formés d'une succession d'images gif normales). Les revues informatiques proposent très souvent des CD comportant des images, icônes et même animations.

Mais l'idéal reste bien sûr de créer soi-même des images. Un certain nombre de programmes existent, comme **Photoshop** ou Paint Shop Pro.



Rien n'est plus désagréable qu'une page WEB prétendant être une source de renseignements mais qui est trop embrouillée parce que désordonnée. Le remède ? Utiliser des listes à puces ou des tableaux ...

V. Les listes et les tableaux



On dit que les prochaines générations seront difficiles à gouverner. Je l'espère bien.

(Alain, Propos sur l'éducation)

Un texte continu n'est pas toujours la forme la plus adaptée à la présentation d'informations. Très souvent, en effet, celles-ci se présentent sous forme d'énumérations ou de données structurées de manière plus complexes. Dans ce cas, les listes et les tableaux sont les éléments de présentation idéaux.

1. Les listes à puces

Bien connues des traitements de textes, les listes à puces se créent sans difficulté en cliquant sur l'icône de liste :



Une puce en forme de disque noir apparaît. Il suffit d'encoder le premier item de la liste. Un Enter fera apparaître une deuxième puce. Et ainsi de suite ... Le dernier Enter de la liste fait encore apparaître une puce : il suffit de cliquer sur l'icône de liste pour la faire disparaître (visuellement, le bouton n'est plus enfoncé). On obtient ainsi pour notre exemple la liste remaniée des céréales :

A screenshot of a Netscape browser window. The title bar says "Home-page de la ferme - Netscape". The menu bar includes File, Edit, View, Go, Bookmarks, Tools, Window, Help. The toolbar includes Back, Forward, Stop, Home, Mail, Radio, My Netscape, Search, and Bookmarks. The main content area displays a page titled "Bienvenue". The text on the page is:

La ferme "[Gédéon et Marie](#)" est située à Oupeye. Ses activités s'étendent non seulement à l'élevage mais aussi à la production céréalière. De plus, par ses [activités pédagogiques](#), elle s'inscrit dans une dynamique culturelle de belle envergure dans le monde agricole.

L'élevage est historiquement la première activité. Disposant de 60 ha de prés, la ferme y élève des vaches laitières et des boeufs à destination de boucherie (les pauvres ;-))

Les cultures sont

- le blé;
- l'avoine;
- le maïs;
- les pommes de terre;

Le code HTML met en évidence les balises de listes :

<P>Les cultures sont :</P>

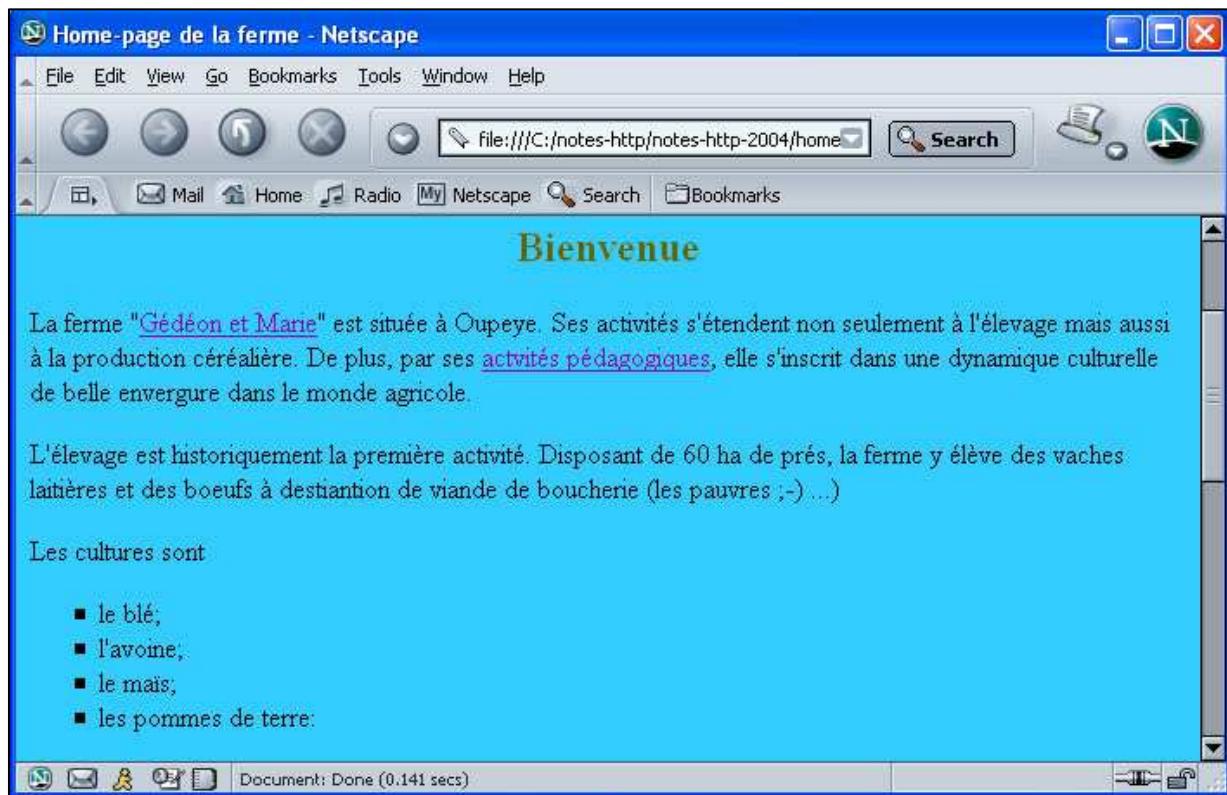
```
<UL>
<LI>le bl&eacute;;</LI>
<LI>l'avoine;</LI>
<LI>le ma&iuml;s;</LI>
<LI>les pommes de terre.</LI>
</UL>
```

Les balises de base sont visiblement **** et **** (pour **Unordered List**) : elles limitent la zone de texte à mettre en liste. Chaque élément de liste est précédé de la balise **** (**List Item**). C'est aussi simple que cela ...

L'attribut **TYPE** du tag **** permet de choisir un format de puce :

```
<UL TYPE=disc>
<UL TYPE=circle>
<UL TYPE=square>
```

fixent bien le format comme on peut le croire ... Ainsi, avec square,

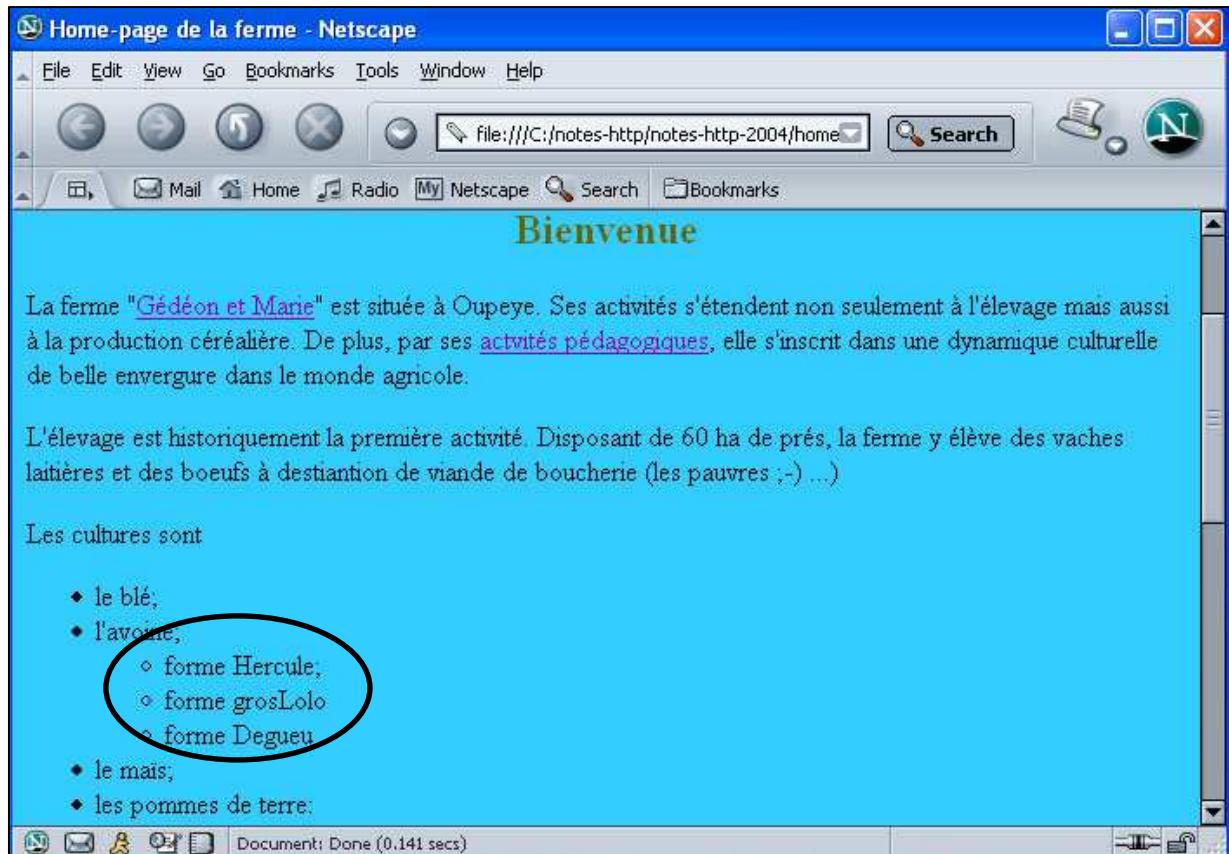


2. Les listes imbriquées

Il est bien sûr possible de définir des listes comme items d'autres listes. Ainsi, pour notre exemple, si l'on souhaite préciser des variétés de céréales, il suffit d'encoder ces variétés à la suite puis de les reculer. Pour ce faire, on sélectionne ces variétés et l'on clique sur l'icône de retrait de paragraphe :



On obtient ainsi :



avec un code HTML prévisible :

```
<ul>
<li> le bl&eacute;;</li>
<li> l'avoine;</li>
<ul>
<li> forme Hercule;</li>
<li> forme grosLolo</li>
<li> forme Degueu</li>
</ul>
<li> le ma&iuml;s;</li>
<li> les pommes de terre:</li>
</ul>
```

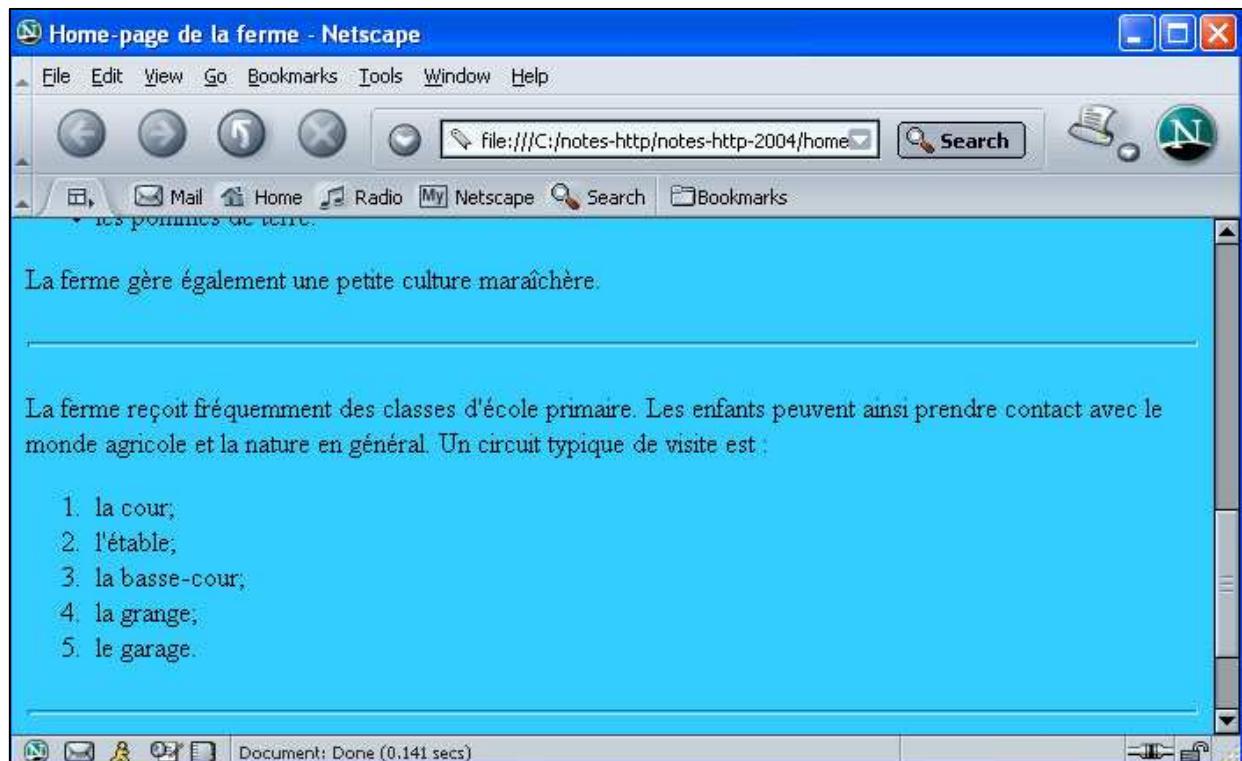
Si l'on n'a pas précisé de type de puce au départ, l'éditeur utilise un autre format de puce par défaut pour les items en retrait. Mais rien n'empêche de changer le format à partir du code HTML.

3. Les listes numérotées

Lorsqu'une notion d'ordre est sous-jacente dans l'énumération, il est préférable de remplacer les puces par des numéros. Il suffit pour cela de cliquer sur l'icône de liste à numéro :



On obtient ainsi pour notre exemple la liste remaniée des étapes du circuit pédagogique :



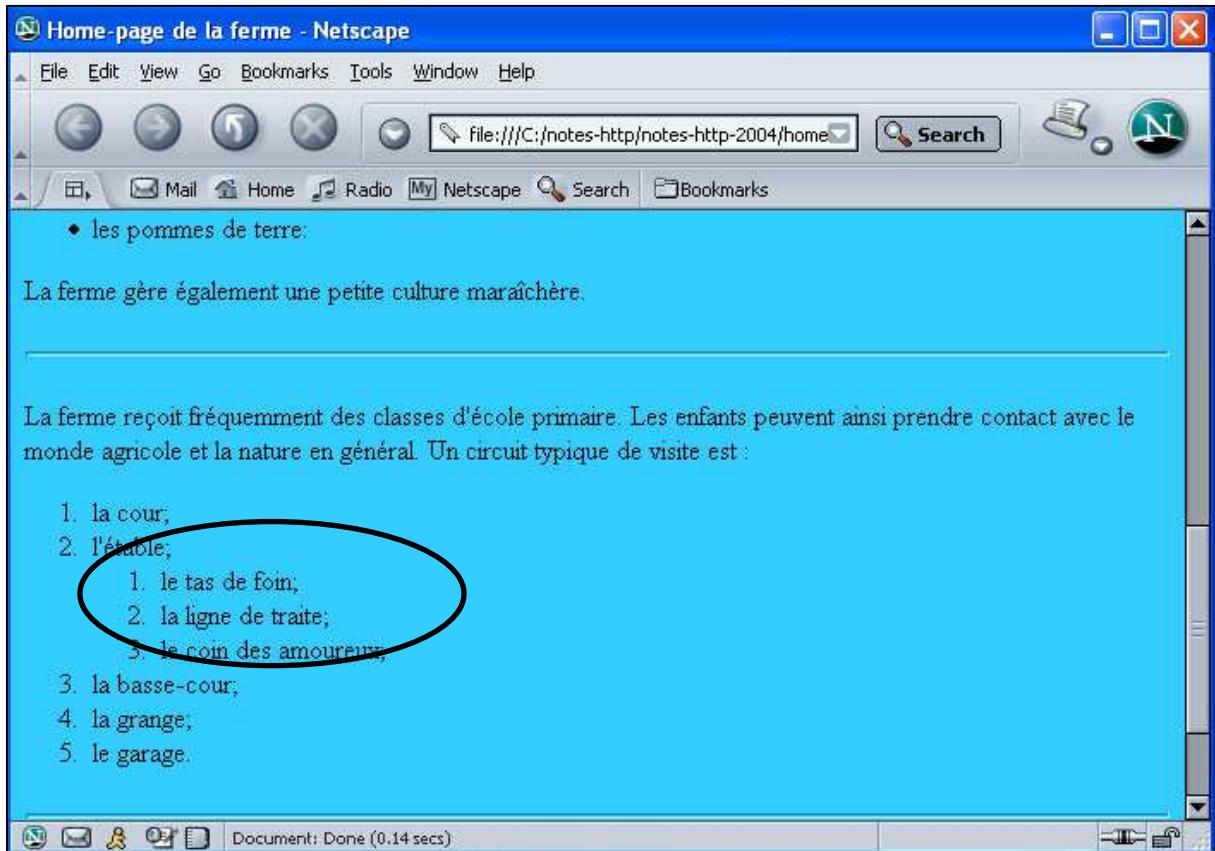
Le code HTML met en évidence les balises de listes numérotées :

```
<OL>
  <LI> la cour;</ LI >
  < LI > l'&eacute;table;</ LI >
  < LI > la basse-cour;</ LI >
  < LI > la grange;</ LI >
  < LI > le garage.</ LI >
</OL>
```

qui sont visiblement **** et **** (pour **Ordered List**) : elles limitent la zone de texte à mettre en liste numérotée.

Les mêmes variations que celles des listes à puces sont possibles :

- ◆ les listes imbriquées :



Mais l'on remarquera que la numérotation reste du même type. On peut utiliser le point suivant :

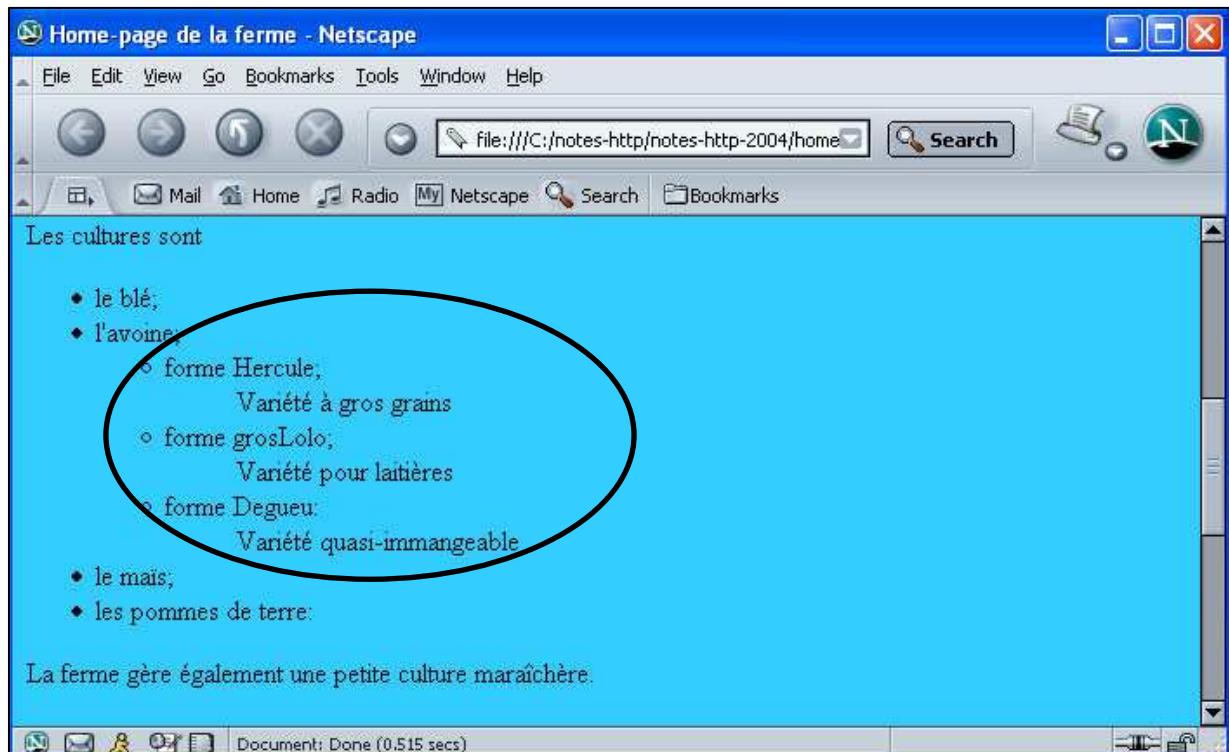
- ◆ les symboles différents au moyen de l'attribut **TYPE** de la balise ****; on peut utiliser :

<OL TYPE=a>
<OL TYPE=I>
<OL TYPE=i>
<OL TYPE=1>

(nous laissons le lecteur sagace deviner les effets ...).

4. Les glossaires

Une variante est la liste de type glossaire, où chaque élément de liste comporte un terme à définir (préfixé de la balise **<DT>** - Definition Term) suivi de sa définition (préfixée de la balise **<DD>** - Definition Data) qui sera placée en dessous et en retrait (dans le menu **Format→List→Term**). Ainsi, la liste des céréales pourrait devenir :



Le code HTML correspondant est :

```
<UL>
<LI>le bl&acute;;</LI>
<LI>l'avoine;</LI>
<UL type=square>
<LI><DT>forme Hercule;<DD>Variété à gros grains</LI>
<LI><DT>forme grosLolo;<DD>Variété pour laitières</LI>
<LI><DT>forme Degueu:<DD>Variété quasi-immangeable</LI>
</UL>
<LI>le ma&iuml;s;</LI>
<LI>les pommes de terre.</LI>
</UL>
```

5. Les menus

La balise `<MENU>`, dont le rôle semble clair, est tombée en désuétude et disparaît avec la norme HTML 3.0. Pourquoi donc ? Parce que créer un menu se fait tout simplement en créant une liste dont chaque item est un lien ... Ainsi, exemple au hasard, voici le menu de l'ancienne home-page de la HEP Rennequin Sualem (souvenir, souvenir ...) :



Evidemment, une liste ne demande qu'à se transformer en tableau ...

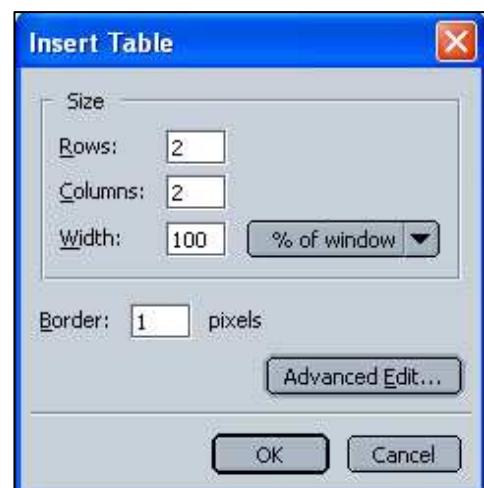
6. Les tableaux de base

Nos habitudes de la pratique du traitement de textes font que les tableaux nous paraissent une nécessité. A l'heure actuelle, tous les navigateurs les reconnaissent.

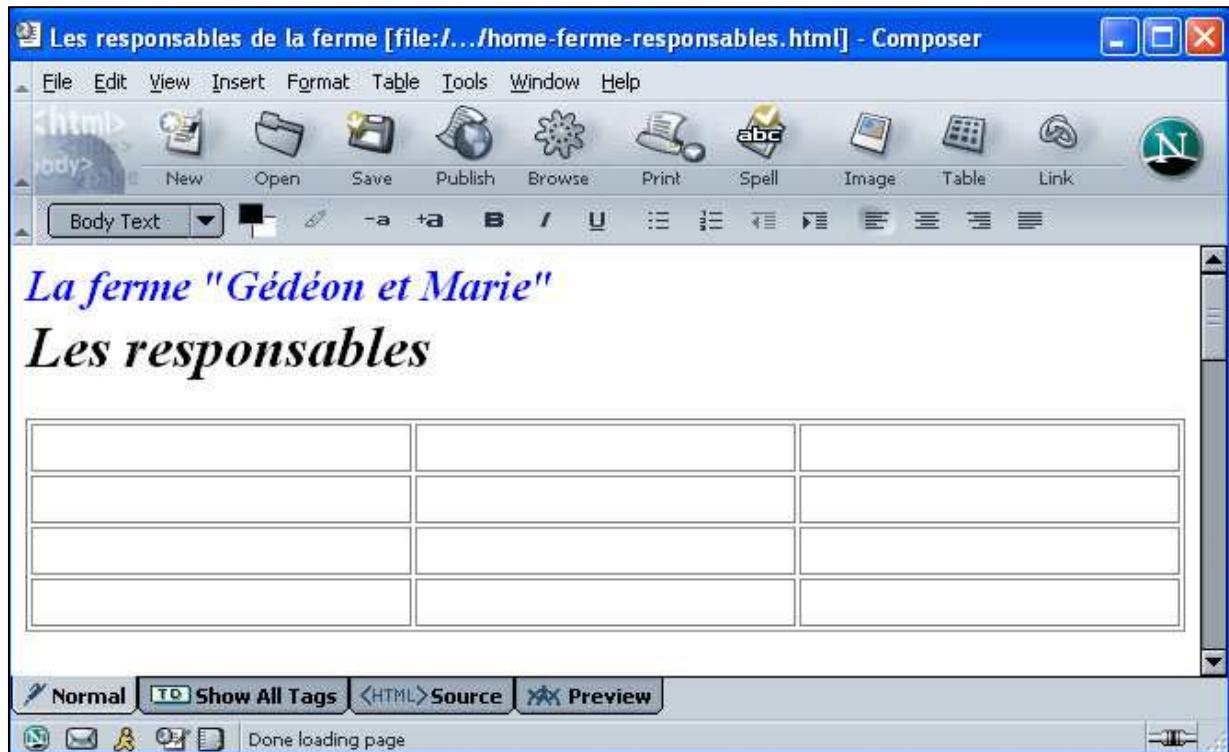
La création d'un tableau classique (c'est-à-dire en forme de quadrillage) débute par un clic sur l'icône des tableaux :



La boîte de dialogue que l'on obtient est très simple :



Dans un premier temps, il suffit de choisir le nombre de lignes [*rows*] et le nombre de colonnes [*columns*]. Par exemple, si l'objectif est de réaliser dans une nouvelle page un tableau comportant les responsables des différents secteurs de la ferme-modèle avec leurs attributions et leurs horaires, on choisira 4 lignes et 3 colonnes. Bien sûr, ce choix pourra être modifié dans la suite, au moyen de l'éditeur graphique ou en écrivant au sein du code HTML. Un OK volontaire donne le résultat suivant :



Il suffit à présent d'encoder du texte dans chaque cellule. Si on réalise l'encodage d'un tableau comme celui-ci¹⁰ :

Christophe DELANTENNE	maître des bœufs	7h30-16h30
Laurence BONNEHUMEUR	directrice financière	9h-15h30
Denys UNIXALAISE	maître des grains	
Claude DUMOLLUSQUE	maître des visites	8h-17h

en plaçant chaque item dans sa cellule, la navigation donne :

¹⁰ la plupart des responsables semblent apprécier Fort-Boyard ;-)



et le code HTML qui a été généré est le suivant :

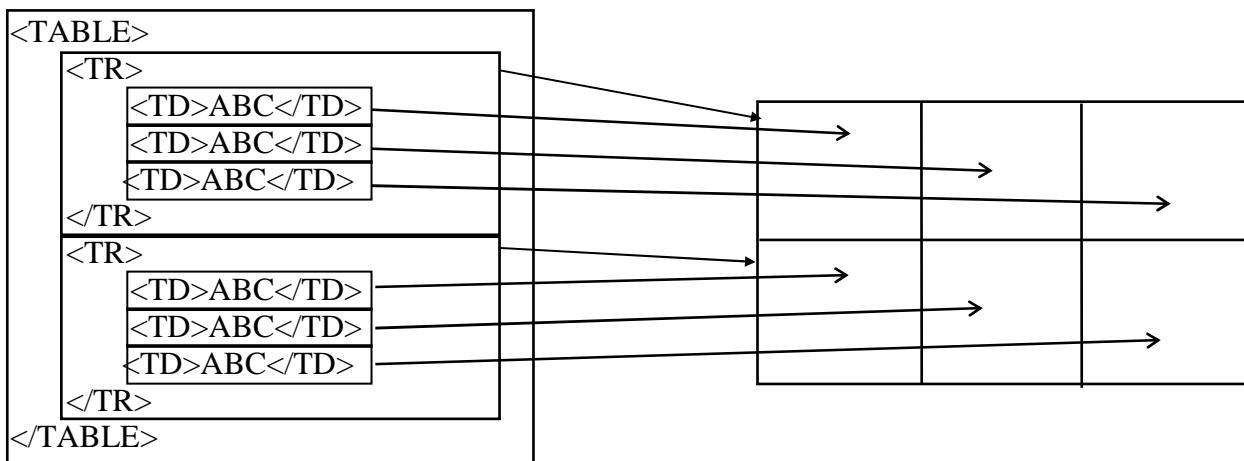
```
<TABLE BORDER COLS=3 WIDTH="100%" >
<tbody>
<tr>
    <td> Christophe DELANTENNE </td>
    <td>ma&icirc;tre des bœufs</td>
    <td>7h30-16h30</td>
</tr>

<tr>
    <td> Laurence BONNEHUMEUR </td>
    <td>directrice financi&egrave;re</td>
    <td>9h-15h30</td>
</tr>

<tr>
    <td> Denys UNIXALAISE </td>
    <td>ma&icirc;tre des grains&nbsp;</td>
    <td></td>
</tr>

<tr>
    <td>Claude DUMOLLUSQUE&nbsp;</td>
    <td>ma&icirc;tre des visites&nbsp;</td>
    <td>8h-17h</td>
</tr>
</tbody>
</TABLE>
```

Celui-ci mérite d'être compris, car il parfois plus facile d'apporter une modification au tableau en rectifiant le code qu'en utilisant les assistants intégrés¹¹. Toutes les données du tableau sont encadrées par les balises **<TABLE>** et **</TABLE>**. Ce tag peut recevoir des attributs, mais ce n'est pas le problème immédiat. Chaque ligne du tableau est délimitée par les marqueurs **<TR>** et **</TR>** (Tag Row). Enfin, les différentes cellules de chaque ligne sont délimitées par les tags **<TD>** et **</TD>** (Tag Data). Ainsi, pour un tableau 2x3 :

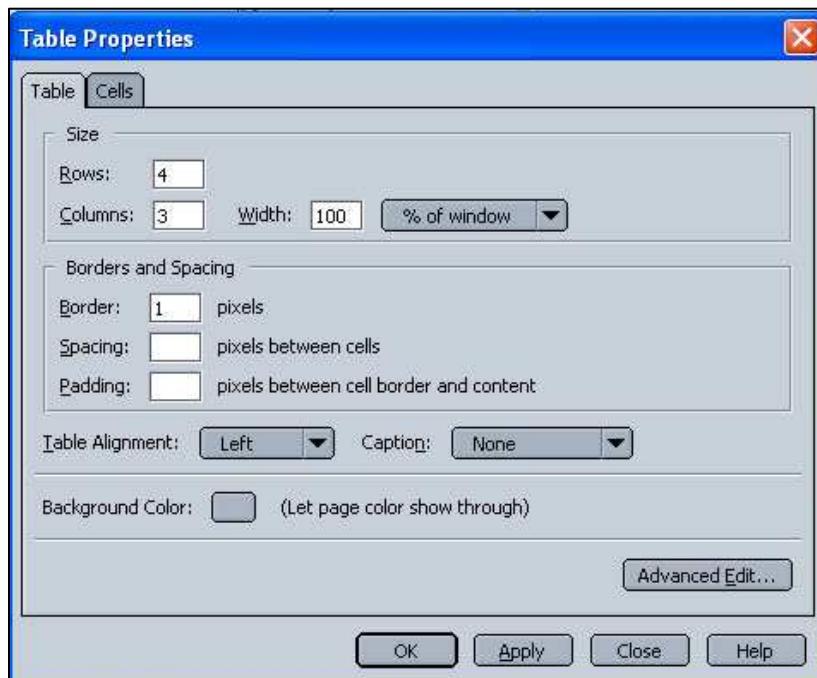


On peut constater que, de cette manière, une case du tableau peut être vide (c'est le cas, pour notre exemple, de l'horaire du maître des grains).

7. Quelques compléments aux tableaux de base

7.1 La mise en page du tableau

A cette trame de base peuvent s'ajouter divers raffinements que l'on peut définir en choisissant dans le menu contextuel obtenu par un clic droit *Table Cell Properties* – ceci nous donne une boîte de dialogue à deux onglets. Le premier



¹¹ Ceci n'est peut-être qu'une question de point de vue.

permet de définir les propriétés générales du tableaux, comme

- ◆ la largeur des bordures [*border*];
- ◆ l'espacement entre les cellules [*spacing*];
- ◆ l'espacement entre le texte et les bords des cellules [*padding*]
- ◆ l'alignement du fond [*alignment*];
- ◆ la couleur de fond [*background color*].

Ceci peut, par exemple, avoir pour effet d'ajouter à la balise <TABLE> l'attribut **BORDER** :

<TABLE BORDER=3 >

et l'attribut **CELLSPACING** :

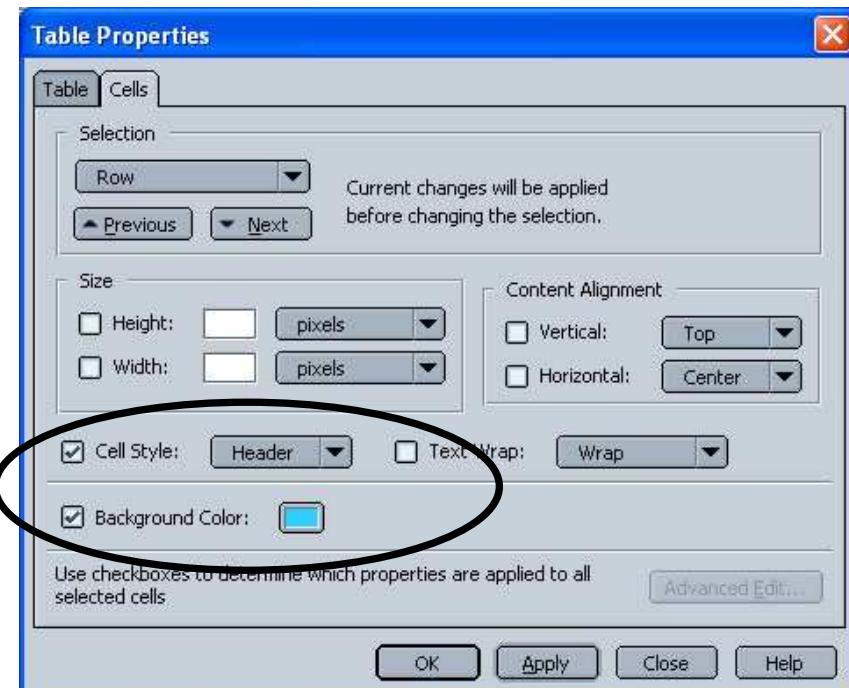
<TABLE BORDER=3 CELLSPACING=2 >

ce qui donnera :



7.2 Les titres de colonnes

On peut aussi définir des titres de colonnes : ils apparaîtront avec un affichage différent des autres cellules si l'on remplace, dans la définition du tableau, les balises <TD> de la première ligne par des balises <TH> (Tag Header). Ceci peut se faire facilement après avoir inséré une ligne de plus au-dessus du tableau, en déterminant le style "header" et, éventuellement, une autre couleur de fond :



Ceci ajoute au code HTML, après la balise <TABLE> et <TBODY> :

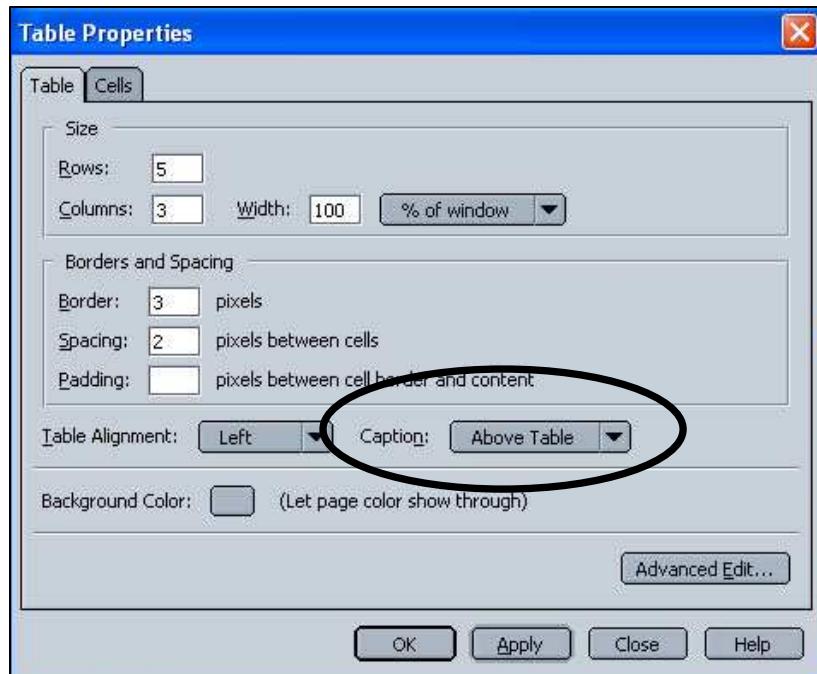
```
<tr>
<th valign="top" bgcolor="#33ccff">Responsable<br>
</th>
<th valign="top" bgcolor="#33ccff">Titre<br>
</th>
<th valign="top" bgcolor="#33ccff">Heures d'ouverture<br>
</th>
</tr>
```

Responsable	Titre	Heures d'ouverture
Christophe DELANTENNE	maître des bœufs	7h30-16h30
Laurence BONNEHUMEUR	directrice financière	9h-15h30
Denys UNDXALAISE	maître des grains	
Claude DUMOLLUSQUE	maître des visites	8h-17h

[Retour à Home-page ferme]

7.3 Le titre général du tableau

On peut forcer l'apparition dans le code HTML de la balise <CAPTION> qui correspond au titre du tableau :



Effectivement, les balises <CAPTION> et </CAPTION> sont apparues juste après la balise <TABLE> : il n'y a plus qu'à encoder le titre en question :

```
<TABLE border="3" cols="3" width="100%" cellspacing="2">
  <CAPTION><big><i><b>Les Ma&icirc;tres</b></i><br></big></CAPTION>
<tbody>
  <tr>
    ...
  </tr>
</tbody>
</table>
```

donne :

Responsable	Titre	Heures d'ouverture
Christophe DELANTENNE	maître des bœufs	7h30-16h30
Laurence BONNEHUMEUR	directrice financière	9h-15h30
Denys UNIKALAISE	maître des grains	
Claude DUMOLLUSQUE	maître des visites	8h-17h

7.4 Modifier le tableau

On peut modifier la structure du tableau par ajout et/ou retrait de lignes, de colonnes ou de cellules en sélectionnant dans le menu habituel obtenu par clic droit depuis la position d'insertion (ou de suppression) :

Table Insert →

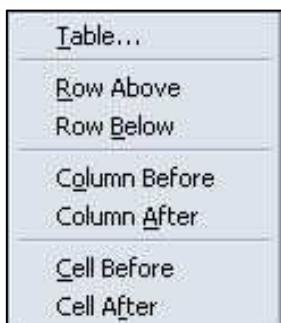
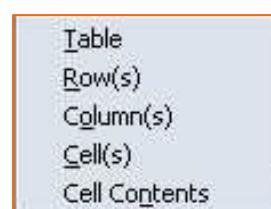


Table Delete →



Evidemment, cela commence à perturber l'ordre cartésien des cellules ! Et justement ...

8. Les tableaux complexes

Il est possible de réaliser des tableaux plus élégants qu'un simple quadrillage, par exemple comme le tableau suivant (productions laitières) :

		Production laitière moyenne (en l)	
		Vaches	Chèvres
Méthode	traditionnelle		
	ferme modèle		

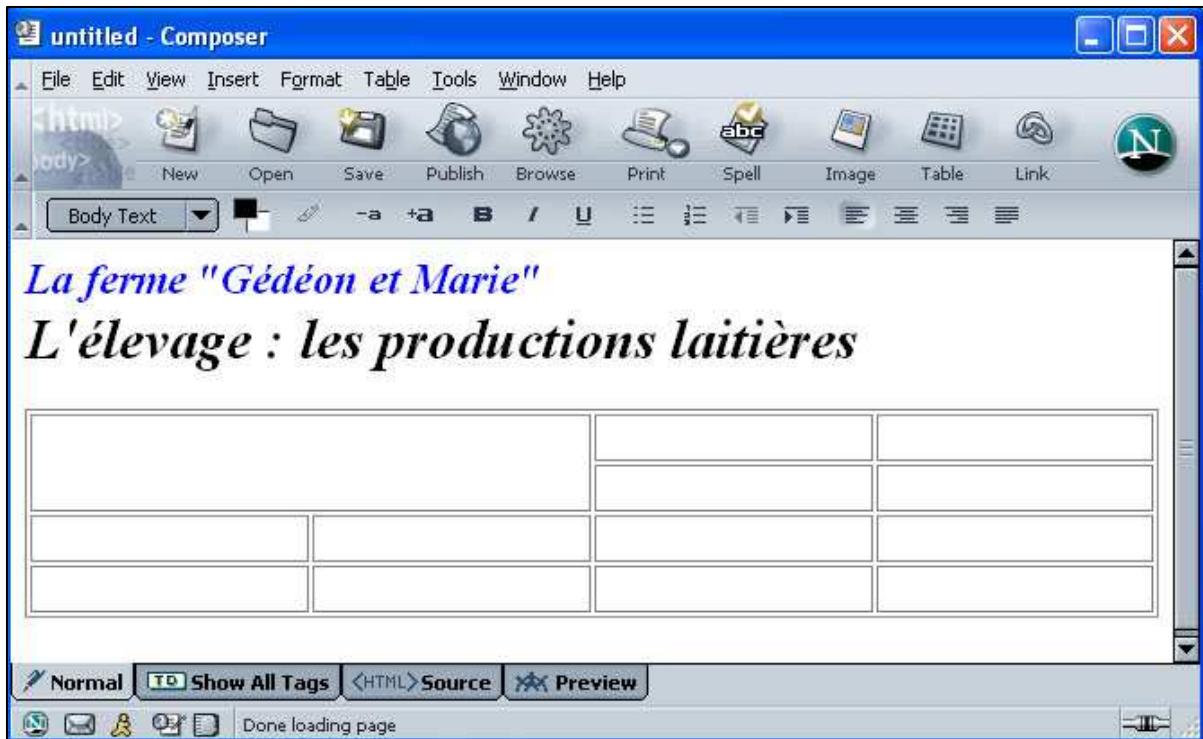
Il faut pour cela utiliser les attributs **ROWSPAN** et **COLSPAN** des balises **<TD>** (et éventuellement **<TH>**) :

- ◆ **ROWSPAN** précise, pour la cellule à laquelle elle s'applique, le nombre de lignes qu'elle recouvre. Ainsi, dans l'exemple ci-dessus, la cellule 'Méthode' a un attribut **ROWSPAN** de 2;
- ◆ **COLSPAN** joue le même rôle, mais pour les colonnes. Toujours pour l'exemple ci-dessus, la cellule vide présente un **COLSPAN** de 2.

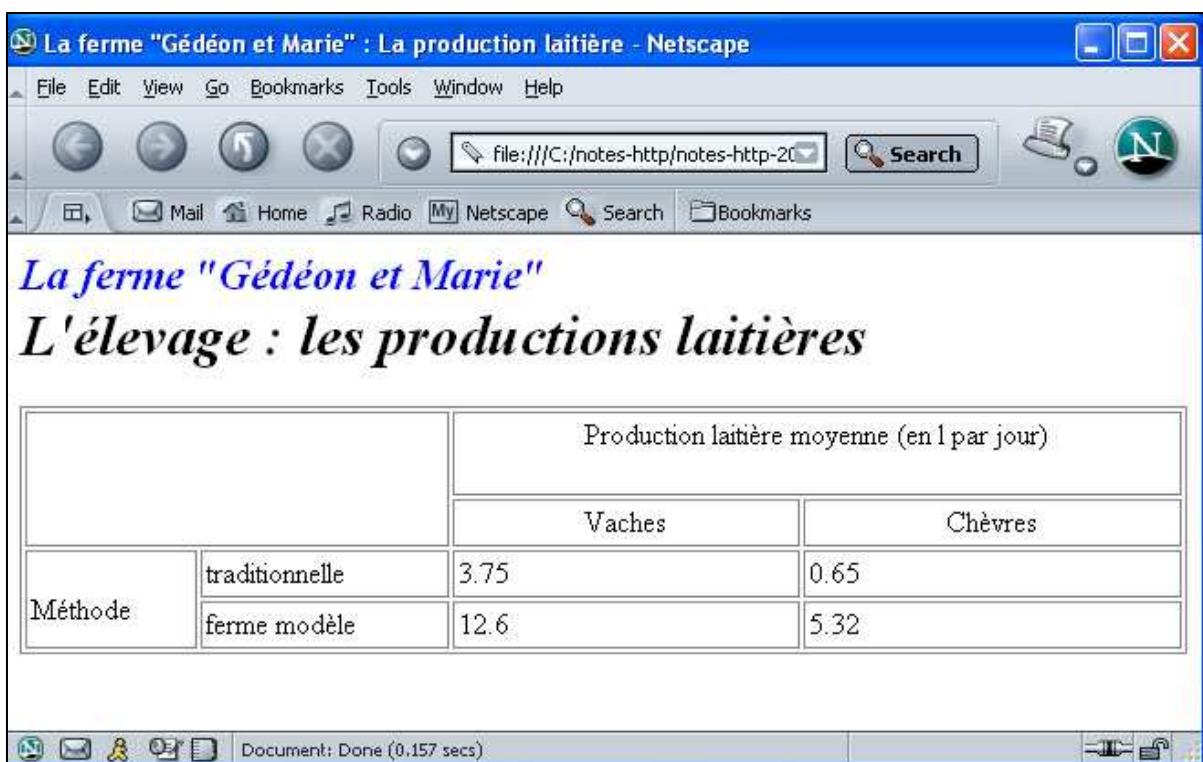
On peut construire un tel tableau au moyen de l'éditeur graphique. On commence par créer un tableau 4x4 :

Plaçons-nous dans la cellule [1,1]. Nous allons ajouter le code HTML complémentaire pour préciser que cette cellule doit recouvrir 2 colonnes. Pour cela, on choisit dans le menu contextuel "Join with Cell to the Right" : c'est fait ! On recommence à la ligne suivante. Reste

à fusionner les deux larges cellules ainsi obtenues : on les sélectionne et on choisit dans le menu contextuel "Join Selected Cells" et le tour est joué.



Evidemment, on aurait pu agir ainsi dès le début avec les 4 cellules. On poursuivant de même, on obtient bien le résultat attendu :



Le code HTML correspondant est :

```
<TABLE cellpadding="2" cellspacing="2" border="1" width="100%">
<TBODY>
<TR>
<TD valign="top" COLSPAN="2" ROWSPAN="2"><br></TD>
<TD valign="top" align="center" ROWSPAN="1" COLSPAN="2">
    Production laiti&egrave;re moyenne (en l par jour)<br></TD>
</TR>
<tr>
<td valign="top" align="center">Vaches<br> </td>
<td valign="top" align="center">Ch&egrave;vres<br> </td>
</tr>
<tr>
<td valign="top" rowspan="2" colspan="1"><br> M&eacute;thode<br> </td>
<td valign="top">traditionnelle<br> </td>
<td valign="top">3.75<br> </td>
<td valign="top">0.65<br> </td>
</tr>
<tr>
<td valign="top">ferme mod&egrave;le<br> </td>
<td valign="top">12.6<br> </td>
<td valign="top">5.32<br> </td>
</tr>
</tbody>
</table>
```



Nous pourrions encore nous pencher sur quelques raffinements complémentaires, mais des cousins des tableaux nous appellent.

En effet, on peut rencontrer des "tableaux" qui sont des "super-tableaux" : leurs composantes sont en fait des fenêtres distinctes. Cela s'appelle des "frames" ...

VI. Les frames



Il y a aujourd'hui une nationalité européenne, comme il y avait, au temps d'Eschyle, de Sophocle et d'Eutipide, une nationalité grecque.

(V. hugo, Les Burgraves)

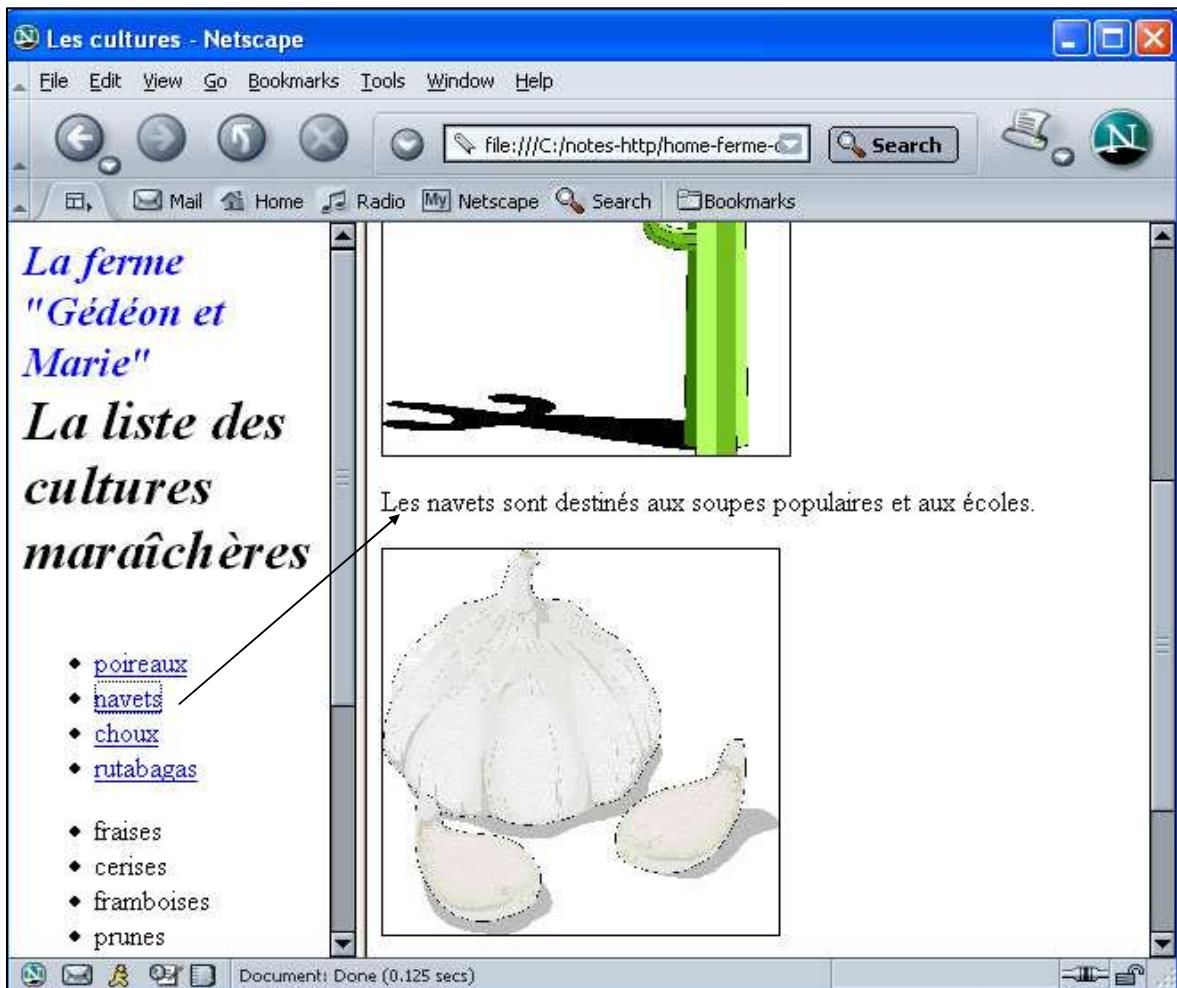
1. L'objectif

L'utilisation des hyperliens peut parfois poser un problème. En effet, le choix dans une liste de références conduit à une autre page, ce qui fait perdre de vue le menu dont on est issu. Bien sûr, un "Back" y fait revenir, mais la vue d'ensemble est néanmoins bel et bien perdue. En fait, on souhaiterait une présentation fenêtrée, avec, par exemple, le menu à gauche et l'article choisi à droite. Pour notre ferme :

A screenshot of a Netscape browser window titled "Les cultures - Netscape". The window has a standard toolbar and menu bar. The left frame contains a menu with items like "La ferme", "Gédéon et Marie", and "La liste des cultures maraîchères". The right frame contains a paragraph about "La ferme 'Gédéon et Marie'" and "Les caractéristiques des cultures maraîchères", followed by a note about the parsnip and a picture of a cactus. A list of vegetables is also present in the left frame.

Ici :

- ◆ un clic sur "poireaux" (à gauche) conduit sur le paragraphe des poireaux (à droite);
- ◆ un clic sur "navets" (à gauche) conduit sur le paragraphe des navets (à droite);
- ◆ etc.



On peut obtenir un tel effet en utilisant les '**frames**' (cadres en français ?). Elles sont à l'heure actuelle fort décriées, d'autant que l'on peut obtenir le même effet avec la balise `<div>` et les feuilles de style (voir chapitre suivant). Néanmoins, elles restent utilisées.

2. La définition d'une page à frames

Une telle page est en fait formée de deux pages présentées dans des fenêtres distinctes. Pour notre exemple, supposons que les deux pages soient `liste-cultures.htm` et `details-cultures.htm`. Supposons-les créées sans les liens. Le code HTML nécessaire pour les deux pages et la page 'principale' ne peut être généré directement avec le Composer de Netscape qui ne sait pas éditer les frames. Il faudra donc éditer la page principale seule – son code est alors celui-ci :

home-ferme-cultures-principale.html

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <meta name="Author" content="Claude Vilvens">
  <meta name="GENERATOR" content="Mozilla/4.7 [fr] (Win98; I) [Netscape]">
  <title>Les cultures</title>
</head>
<FRAMESET cols="30%, 70%" rows="100%">
```

```
<FRAME SRC="home-ferme-liste-cultures.html" NAME="gauche" = "scrolling="auto">
<FRAME SRC="home-ferme-detail-cultures.html" NAME="droite" scrolling="auto">
</FRAMESET>

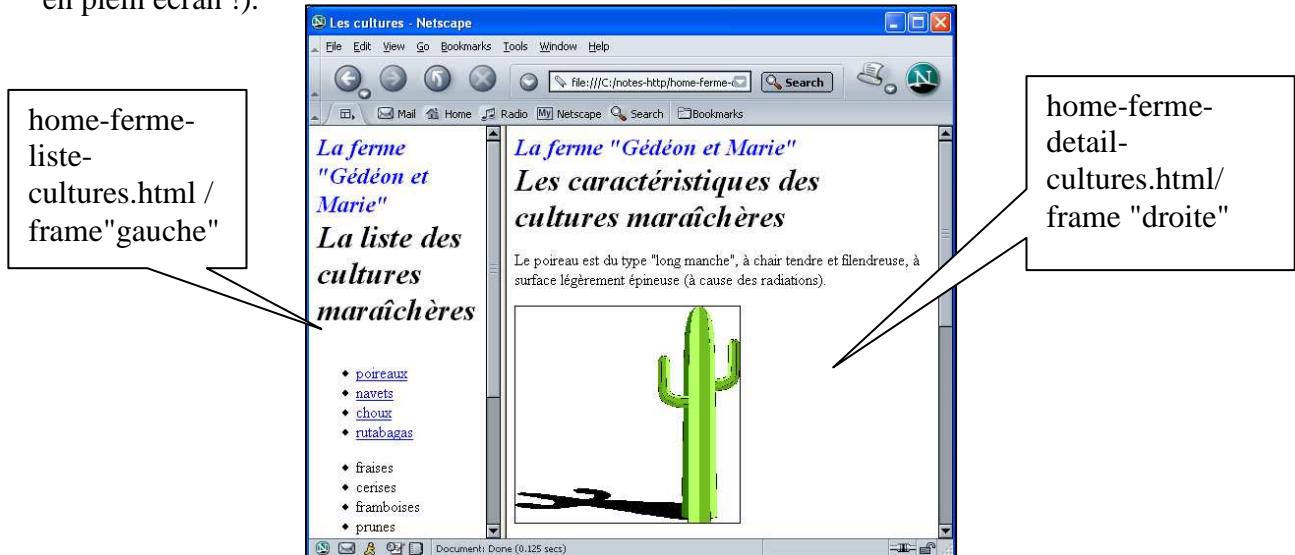
<body>
</body>
</html>
```

On peut constater qu'une structure supplémentaire s'est glissée entre les blocs `<HEAD>` et `<BODY>` : il s'agit du bloc délimité par les balises `<FRAMESET>` et `</FRAMESET>`. C'est bien sûr dans ce bloc que l'on définira les différentes fenêtres. Le tag `<FRAMESET>` admet des attributs **COLS** et **ROWS** dont le rôle est de fixer la taille relative des différents cadres :

- ◆ **COLS** fixe la répartition des colonnes (ici, 30% pour la fenêtre de gauche, 70% pour la fenêtre de droite);
- ◆ **ROWS** fixe la répartition des lignes (ici, chacune occupe la totalité de l'espace disponible).

Chaque cadre est défini par une balise `<FRAME>` qui nécessite au moins deux attributs pour être fonctionnelle :

- ◆ `<SRC>` réclame le nom de la page que le cadre affichera;
- ◆ `<NAME>` permet de donner un nom au cadre, ce qui sera indispensable dans la suite pour le désigner par une référence (une référence à la page conduirait tout simplement à la page en plein écran !).

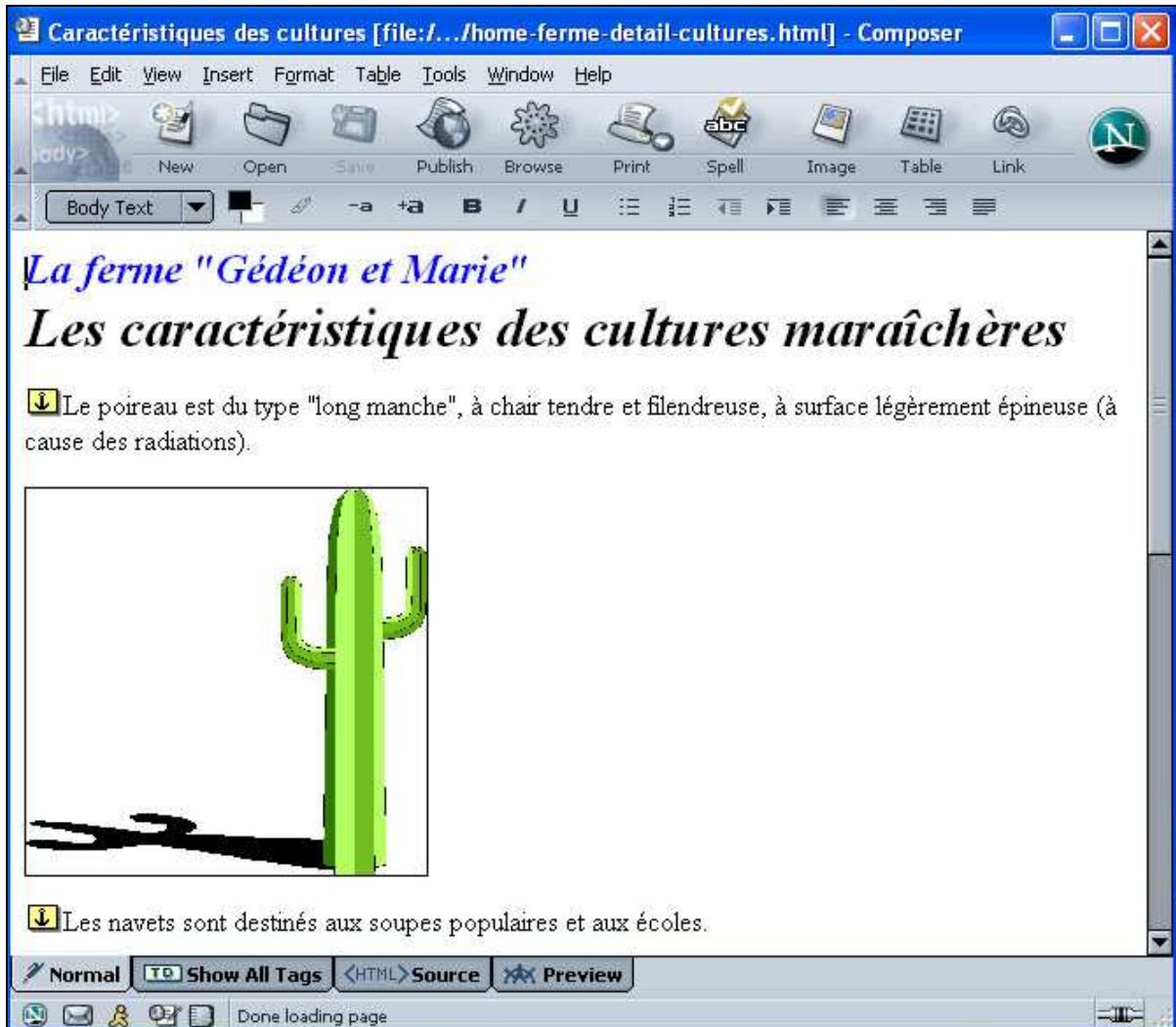


La balise de scrolling n'est pas absolument nécessaire mais est bien pratique.

A remarquer qu'une fois ce texte encodé, rien n'est visible dans l'éditeur graphique (la page semble en fait vide !).

3. Le document cible

Comme l'objectif est de se déplacer au sein du document de droite, il convient d'y placer des signets (anchors) pour permettre ce déplacement. Pour notre exemple, la page home-ferme-detail-cultures.html (incomplète) aura l'aspect suivant :



et le code suivant :

home-ferme-detail-cultures.html

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <meta name="Author" content="Claude Vilvens">
  <meta name="GENERATOR" content="Mozilla/4.7 [fr] (Win98; I) [Netscape]">
  <title>Caract&eacute;ristiques des cultures</title>
</head>

<body>
<b><i><font color="#0000FF"><font size=+2>La
ferme "G&eacute;d&eacute;on et Marie"</font></font></i></b>
```

```
<br><b><i><font size=+3>Les caract&eacute;ristiques des cultures  
mara&icirc;ch&egrave;res</font></i></b>  
  
<p><a NAME="poireau"></a>Le poireau est dy tipe "long manche", &agrave;  
chair tendre et filendreuse, &agrave; surface l&eacute;g&egrave;rement  
&eacute;pineuse (&agrave; cause des radiations).  
<p><img SRC="CACTUS.gif" BORDER=1 height=220 width=228>  
<p><a NAME="navet"></a>Les navets sont destin&eacute;s aux soupes populaires  
et aux &eacute;coles.  
<p><img SRC="NAVET.gif" BORDER=1 height=220 width=222>  
  
<p><a NAME="choux"></a>  
  
<p><a NAME="rutabagas"></a>  
  
<p>[<a href="home-ferme.html">Retour &agrave; Home-page ferme</a>]  
<br>&nbsp;  
</body>  
</html>
```

On peut constater que la présence de quatre signets définis par une balise `<A NAME>`.

4. Le document des références

Le document de gauche contient un menu en forme de liste de références. On crée ces liens comme d'habitude. Pour notre exemple :



Rien de bien spectaculaire : tout est caché dans le code !

liste-cultures.htm

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="Author" content="Claude Vilvens">
<meta name="GENERATOR" content="Mozilla/4.7 [fr] (Win98; I) [Netscape]">
<title>Liste des cultures</title>
</head>
<body>
<b><i><font color="#0000FF"><font size=+2>La ferme "G&eacute;d&eacute;on et Marie"</font></i></b>
<br><b><i><font size=+3>La liste des cultures mara&icirc;ch&egrave;res</font></i></b>
<br>&nbsp;

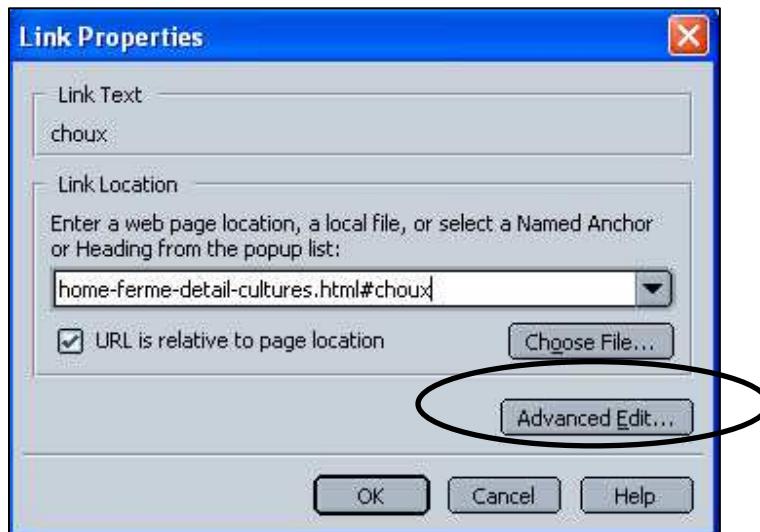
<ul>
<li> <A HREF="home-ferme-detail-cultures.html#poireau" TARGET="droite">poireaux</a></li>
<li> <a href="home-ferme-detail-cultures.html#navet" target="droite">navets</a></li>
<li> <a href="home-ferme-detail-cultures.html#choux" target="droite">choux</a></li>
<li> <a href="home-ferme-detail-cultures.html#rutabagas" target="droite">rutabagas</a></li>
</ul>
<ul>
<li> fraises</li>
<li> cerises</li>
<li> framboises</li>
<li> prunes</li>
</ul>

<p><br><a href="home-ferme.html">Retour &agrave; Home-page ferme</a>
<br>&nbsp;
<br>&nbsp;
<br>&nbsp;
</body>
</html>
```

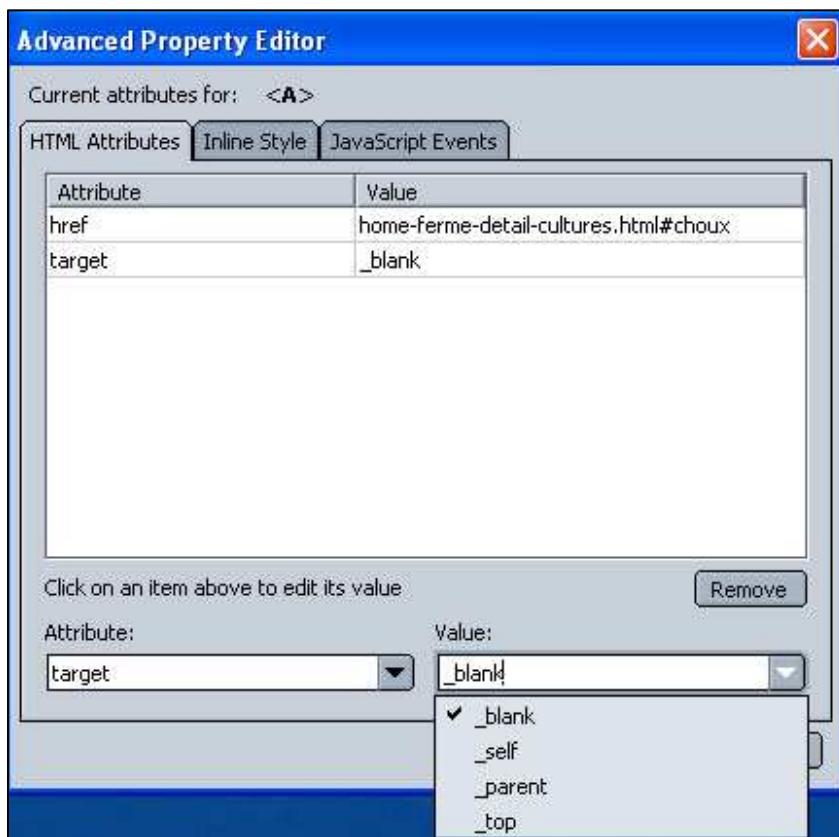
La nouveauté se trouve dans les balises de références. Celles-ci sont du type :

poireaux

La référence désigne bien la page et le signet ad hoc. Mais l'attribut **TARGET** précise dans quelle frame elle doit apparaître, soit ici celle qui se nomme "droite". Sans ce tag, toute référence se ferait dans le cadre courant ... On peut ajouter ces attributs TARGET dans le code HTML lui-même. Mais on peut aussi le faire au moment où l'on crée les liens. Ainsi, pour les choux :



un clic sur le bouton "Advanced Edit ..." fournit une fenêtre d'encodage de l'attribut complémentaire :

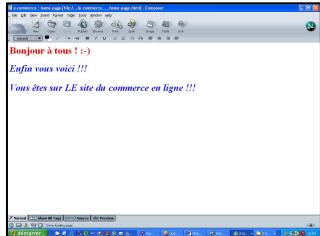


On remarquera les valeurs prédéfinies pour l'attribut TARGET, comme par exemple "**_top**" qui permet d'agrandir la page à la fenêtre entière du navigateur, "**_blank**" pour créer une nouvelle fenêtre, "**_self**" pour afficher dans le même cadre



Jusqu'à présent, la présentation de chaque page est indépendante des autres pages d'un même site. Si un caractère individualiste se satisfait sans problèmes de cet état de fait, un caractère plus pragmatique y trouvera rapidement à redire. Pourquoi ?

VII. Les feuilles de style CSS



*J'estime par-dessus tout d'abord le style,
et ensuite le vrai.*

(G. Flaubert, Correspondance)

1. Une plus grande souplesse

Supposons qu'un site WEB ait été développé et comporte un bon nombre de pages. Le choix du "look" de la présentation a été pensé en son temps en termes de polices de caractères, de couleurs, de tailles, de position des divers éléments, d'images des différents tags usuels. Fort bien, mais un nouveau directeur prend ses fonctions et réclame des changements en ces mêmes termes de couleurs, polices, etc. Va-t-il falloir apporter toutes les modifications nécessaires dans chaque page une par une ? Non ... si on a utilisé des feuilles de style ;-)

On l'aura compris, la technique des feuilles de style **CSS** (Cascading Style Style) consiste à

mémoriser en un seul point les attributs de présentation des différentes balises utilisées dans un certain nombre de pages WEB

(soit au début d'une page, soit, surtout, dans un fichier indépendant d'extension **css**) de manière à ce que tout changement éventuel ne doive s'effectuer qu'en ce seul point, provoquant ainsi une mise à jour automatique de toutes les pages utilisant ces balises.

Les spécifications CSS ont été définies par le W3C de manière à établir une norme de représentation de ces attributs. Evidemment, le concept étant relativement neuf, les anciens browsers ne sont pas capables d'utiliser ces feuilles de style – ceci ne provoquera pas une erreur, mais les styles ainsi définis ne seront pas appliqués ...

2. Les règles de style en interne

Afin d'appréhender les bases du mécanisme, définissons quelques styles personnalisés au sein même du document – on parle encore de "définition interne d'une feuille de style". Celle-ci doit se trouver dans la zone **<head>** du document HTML et est définie dans une balise **<STYLE>** dont une syntaxe courante est :

```
<style type="text/CSS" lang="initiales_langue" media="type_media_concerné">
    balise { propriété:valeur; propriété:valeur; ... }
    balise { propriété:valeur; propriété:valeur; ... }
    ...
</style>
```

où

- ◆ *initiales_langue* peut être, bien sûr : FR (français), EN (anglais), EN-US (anglais américain), DE (allemand), etc
- ◆ *type_media_concerné* peut être
 - screen : écran d'ordinateur (valeur par défaut – logique ...);
 - tv : écran de télévision;
 - projection : à votre avis ?
 - print : pour tout ce qui touche à la publication, y compris les prévisualisations;
 - aural : tout ce qui est auditif;
 - all : tout type de media.

On peut imaginer par exemple de personnaliser les balises de formatage de texte H1 et H2 et même de créer une nouvelle balise, disons HACCROCHE (pour "titre accrocheur"). Cela pourrait donner :

e-commerce__home-page.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>e-commerce : home-page</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">

<style type="text/CSS" lang="FR" media="all">
  h1 { color: red; font-family: times; }
  haccroche { color:blue; center; font-family: times; font-style:italic; font-weight: bold;
             font-size: xx-large; text-decoration:blink; }
  h2{ color:blue; center; font-family: times; font-style:italic; font-weight: bold; font-size:
      xx-large; }
</style>
</head>

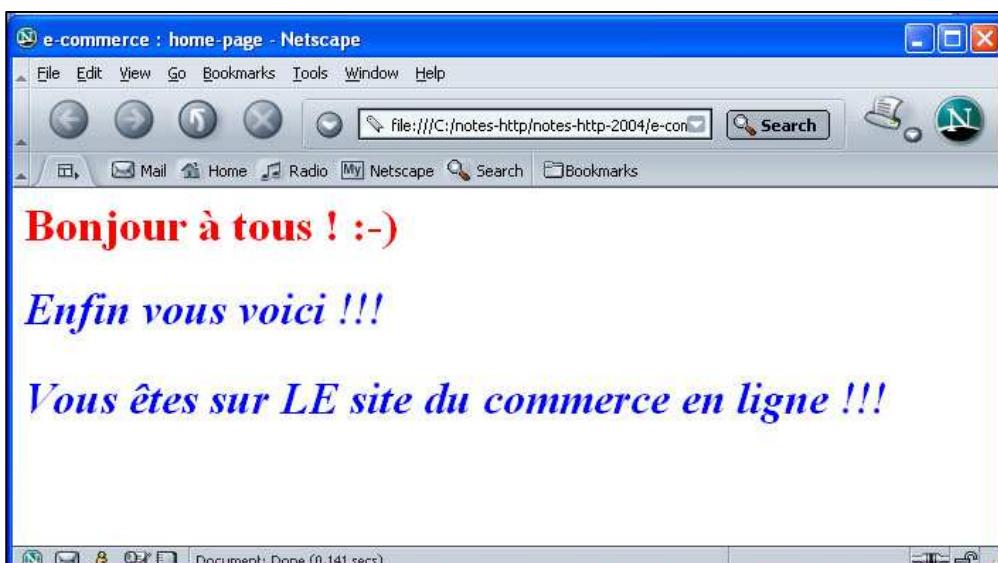
<body>
<h1>Bonjour &agrave; tous ! :-)</h1>
<haccroche> Enfin vous voici !!! </haccroche>
<h2> Vous &ecirc;tes sur LE site du commerce en ligne !!! </h2>
<br>
</body>
</html>
```

On a utilisé ici quelques unes des propriétés les plus courantes :

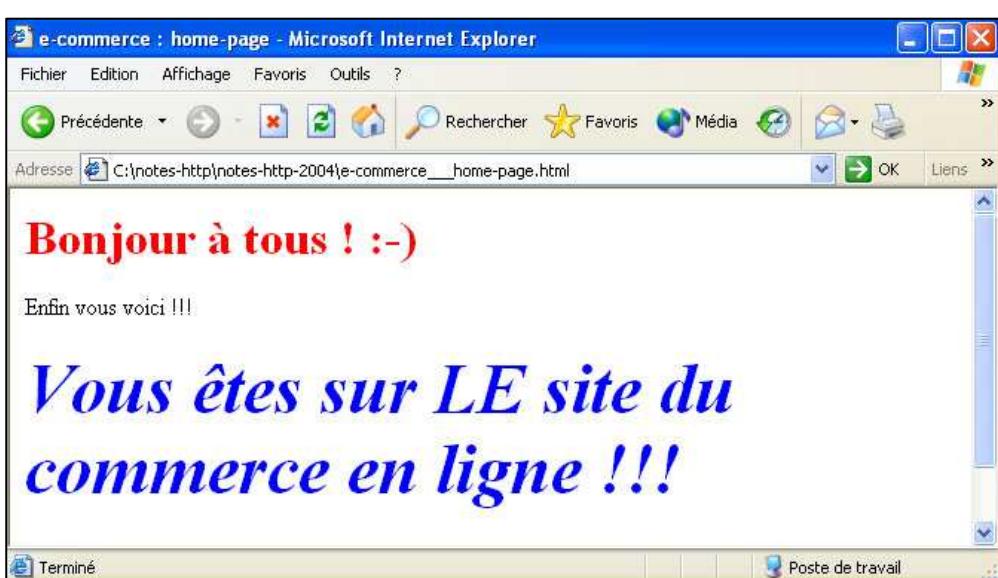
- ◆ **color** : se décline aussi en plusieurs versions c'est-à-dire
 - initiales de couleur comme red, green, blue, etc – ex: color:blue;
 - codage RGB : en hexadécimal, donc précédé de "#" – ex: color:#FF00AA;
 - pourcentage RGB : avec le mot "rgb" – ex: color:rgb(100%, 0%, 80%);
- ◆ **font-family**: sans-serif, cursive, fantasy, times, arial, etc;

- ◆ **font-size** : se décline aussi en plusieurs versions c'est-à-dire
 - taille réelle : nombre décimal suivi des initiales de l'unité (in, cm, mm, px, pt)
– ex: 1.186cm;
 - pourcentage par rapport à la police courante – ex: 125%;
 - tailles prédéfinies : en utilisant les valeurs du browser connues sous les noms xx-small, x-small, small, medium, large, x-large, xx-large.
- ◆ **font-style** : normal (par défaut), italic ou oblique;
- ◆ **font-weight** : normal (par défaut), bold, bolder, lighter;
- ◆ **text-decoration** : none (par défaut), underline, overline, line-through, blink
- ◆ **text-transform** : none (par défaut), capitalize, uppercase, lowercase;
- ◆ **text-align** : left, right, center, justify.

Et il en existe encore bien d'autres ... En visualisant cette page avec Netscape, cela donne :



Cela ne se voit pas bien sur du papier ;), mais la phrase "Enfin vous voici" est bien bleue et elle clignote ... Par contre, avec Internet Explorer :



Autrement dit, ce browser ne reconnaît pas les balises exotiques !

3. Les règles de style dans un fichier externe

Evidemment, si cet exemple nous permet de saisir ce que l'on entend par "style", il est par contre sans intérêt réel par rapport à la problématique de la souplesse des pages, puisque, précisément, chacune contient la définition des styles qu'elle emploie. Tout l'intérêt est évidemment de placer les styles personnalisés dans un fichier bien précis, d'extension **css**, qui servira de référence à l'ensemble des pages du site on parle encore de "définition externe d'une feuille de style". On peut donc imaginer ceci (édition avec un simple bloc-notes) :

e-commerce__home-page.css

```
h1 { color: red; font-family: times; }  
haccroche { color:blue; center; font-family: times; font-style:italic; font-weight: bold; font-size: xx-large; text-decoration:blink }  
h2 { color:blue; center; font-family: times; font-style:italic; font-weight: bold; font-size: xx-large }
```

La page HTML qui utilise cette feuille de style le signifiera dans sa zone <head> au moyen d'une balise <LINK> dont la syntaxe courante est :

```
<link href="chemin_fichier_css" type="text/css" rel="type_relation"  
      media="type_media_concerné" />
```

où

- ◆ *chemin_fichier_css* est bien sûr le chemin du fichier définissant la feuille de style;
- ◆ *type_relation* définit le rapport entre le fichier HTML et le fichier CSS – dans notre cas, ce sera "stylesheet", mais cela pourrait être "glossary", "index", etc.

Donc, pour notre exemple, si la feuille de style a été placée dans un sous-répertoire "styles" du répertoire des pages, cela donnera :

e-commerce__home-page_aveccss.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  
<html>  
  <head>  
    <title>e-commerce : home-page</title>  
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">  
  
    <link href="styles/e-commerce__home-page.css" type="text/css" rel="stylesheet"  
          media="screen" />  
  
  </head>  
  
  <body>  
    <h1>Bonjour &agrave; tous ! :-)</h1>  
    <haccroche> Enfin vous voici !!! </haccroche>  
    <h2> Vous voici sur LE site du commerce en ligne !!! </h2><br>  
  </body>  
</html>
```

Le résultat visuel est identique à celui obtenu avec la méthode interne, y compris avec un serveur WEB qui n'est pas local.

4. Les classes de style

Pas de panique ! Il ne s'agit pas ici de POO ;-), mais de classification. En fait, il est possible de d'associer à un intitulé de balise donné (H1 par exemple) une série de styles différents mais voisins. Ainsi, par exemple :

```
e-commerce ____ home-page avec classes.css
h1.classique { color:blue; font-family:times; }
h1.grave { color:##FF00FFF; font-family: times; font-size:xx-large; }
haccroche { color:blue; center; font-family: times; font-style:italic; font-weight: bold; font-size: xx-large; text-decoration:blink }
h2 { color:blue; center; font-family: times; font-style:italic; font-weight: bold; font-size: xx-large }
```

En quelque sorte, h1 est une "classe" dont "classique" et "grave" sont les membres - mouais ... mais "classe" désigne ici plutôt les membres ☺. Un page HTML utilisant la balise H1 pourra à présent spécifier quel style elle en utilise en ajoutant un attribut "class" :

```
e-commerce ____ home-page avec classes.html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>e-commerce : home-page</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">

<link href="styles/e-commerce ____ home-page%20avec%20classes.css"
      type="text/css" rel="stylesheet" media="screen">
</head>
<body>

<h1 class="classique">Bonjour &agrave; tous ! :-)</h1>
<h1 class="grave">Je disais : Bonjour &agrave; tous ! :-)</h1>
<haccroche> Enfin vous voici !!! </haccroche>
<h2> Vous voici sur LE site du commerce en ligne !!! </h2>

</body>
</html>
```

Le résultat est conforme à ce que l'on attend :



En particulier, il est possible de définir des classes universelles, c'est-à-dire non attachées à un tag particulier et utilisables avec toute balise compatible. Ainsi, si nous définissons dans le fichier css des classes "titre" et "important" :

e-commerce ____ home-page avec classes.css (2)

```
h1.classique { color:blue; font-family:times; }
h1.grave { color:##FF00FFF; font-family: times; font-size:xx-large; }
.titre { font-weight: bold; color:black; font-size:xx-large; text-decoration:underline; }
.important { font-weight: bold; color:red; font-size:x-large; }
haccroche ...
```

nous pouvons les utiliser, par exemple, avec des balises H6, <TBODY> ou <TD> :

e-commerce ____ home-page avec classes.html (2)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>e-commerce : home-page</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">

<link href="styles/e-commerce ____ home-page%20avec%20classes.css"
      type="text/css" rel="stylesheet" media="screen">
</head>
<body>

<h6 class="titre"> --- Page d'accueil --- </h6>

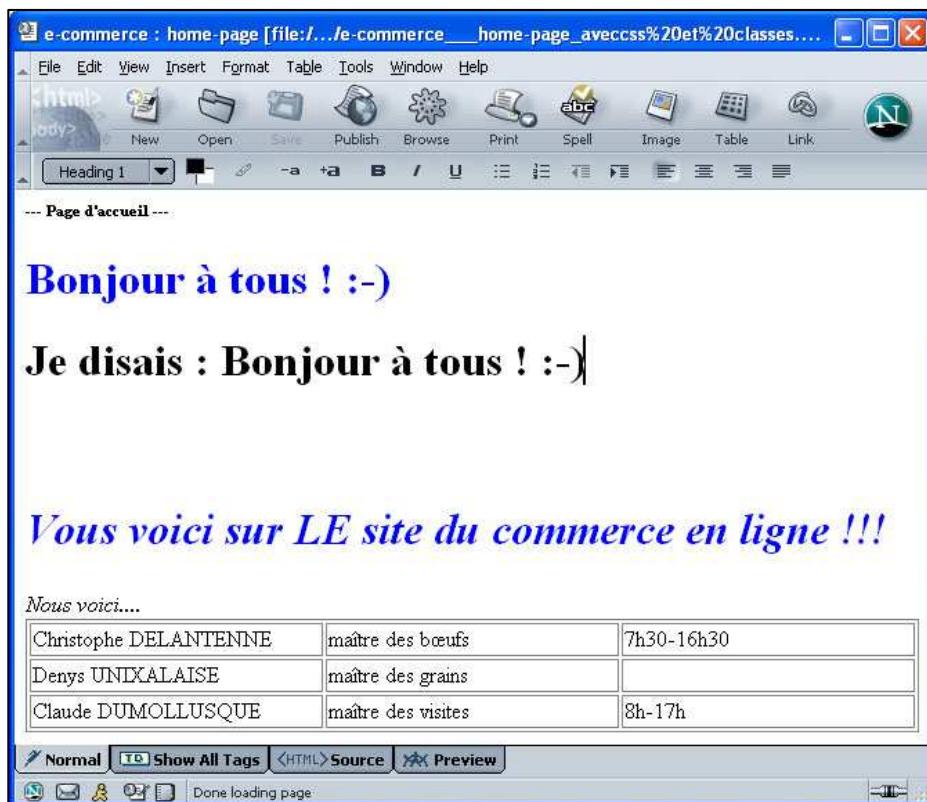
<h1 class="classique">Bonjour &agrave; tous ! :-)</h1>
<h1 class="grave">Je disais : Bonjour &agrave; tous ! :-)</h1>
<haccroche> Enfin vous voici !!! </haccroche>
<h2> Vous voici sur LE site du commerce en ligne !!! </h2>
```

```
<i>Nous voici....</i><br>

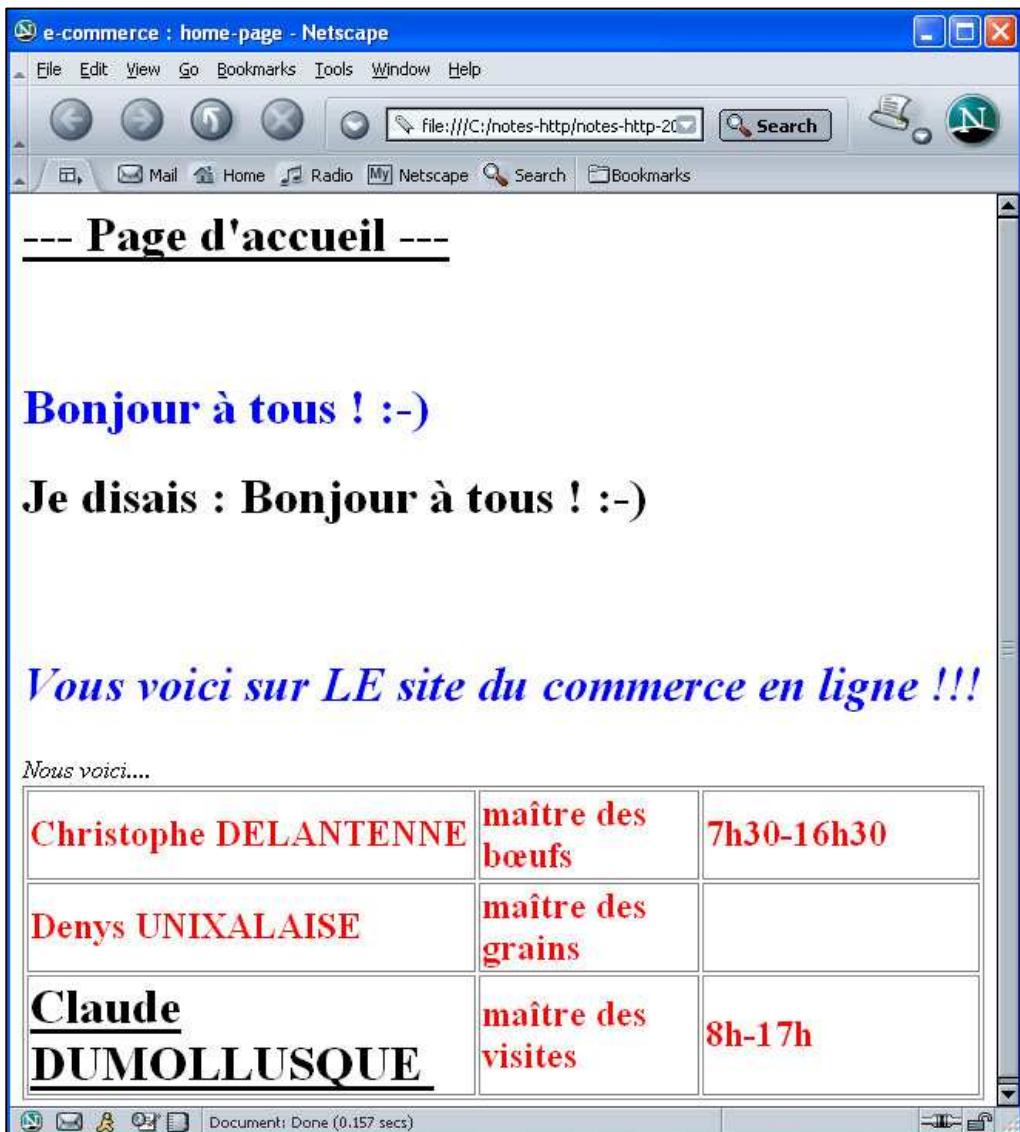
<table border="1" cols="3" width="100%">
  <tbody class="important">
    <tr>
      <td> Christophe DELANTENNE </td>
      <td>maître des bœufs</td>
      <td>7h30-16h30</td>
    </tr>
    <tr>
      <td> Denys UNIXALAISE </td>
      <td>maître des grains&nbsp;</td>
      <td><br> </td>
    </tr>
    <tr>
      <td class="titre">Claude DUMOLLUSQUE&nbsp;</td>
      <td>maître des visites&nbsp;</td>
      <td>8h-17h</td>
    </tr>
  </tbody>
</table>

</body>
</html>
```

Si l'éditeur HTML utilisé ne semble guère intéressé par nos tags remaniés :



il n'en est pas de même du browser (qu'il s'agisse de Netscape ou d'Internet Explorer ou autre) :



5. Les styles et les balises div

La balise **<div>** (comme d'ailleurs sa cousine la balise ****, moins générale) a été créée en conjonction avec les feuilles de style CSS : elle permet d'appliquer un style particulier à un bloc quelconque au sein d'une page HTML. A priori, la balise **<div>** permet de définir une zone quelconque dans une page HTML, zone pouvant couvrir plusieurs paragraphes : comme cette zone ne correspond à aucune élément de structure précis au sein de la page, on dit encore que **<div> est une balise sans contenu sémantique**. Il n'y a lors plus qu'à y appliquer un style particulier.

Illustrons ceci avec un problème classique : la structuration d'une page en bandeau-contenu-pied de page ...

5.1 Une mise en page à 3 zones

Nous définissons tout d'abord une feuille de style CSS décrivant la mise en page :

structure-page.css

```
div
{
    text-align:center;
}

div#bandeau
{
    width:1000px;
    height:40px;
    background-color:#FFCC99;
    font-size:large;
}
div#contenu
{
    width:1000px;
    height:400px;
    background-color:#CCFFFF;
}
div#piedpage
{
    width:1000px;
    height:30px;
    background-color:#FFCC33;
}
```

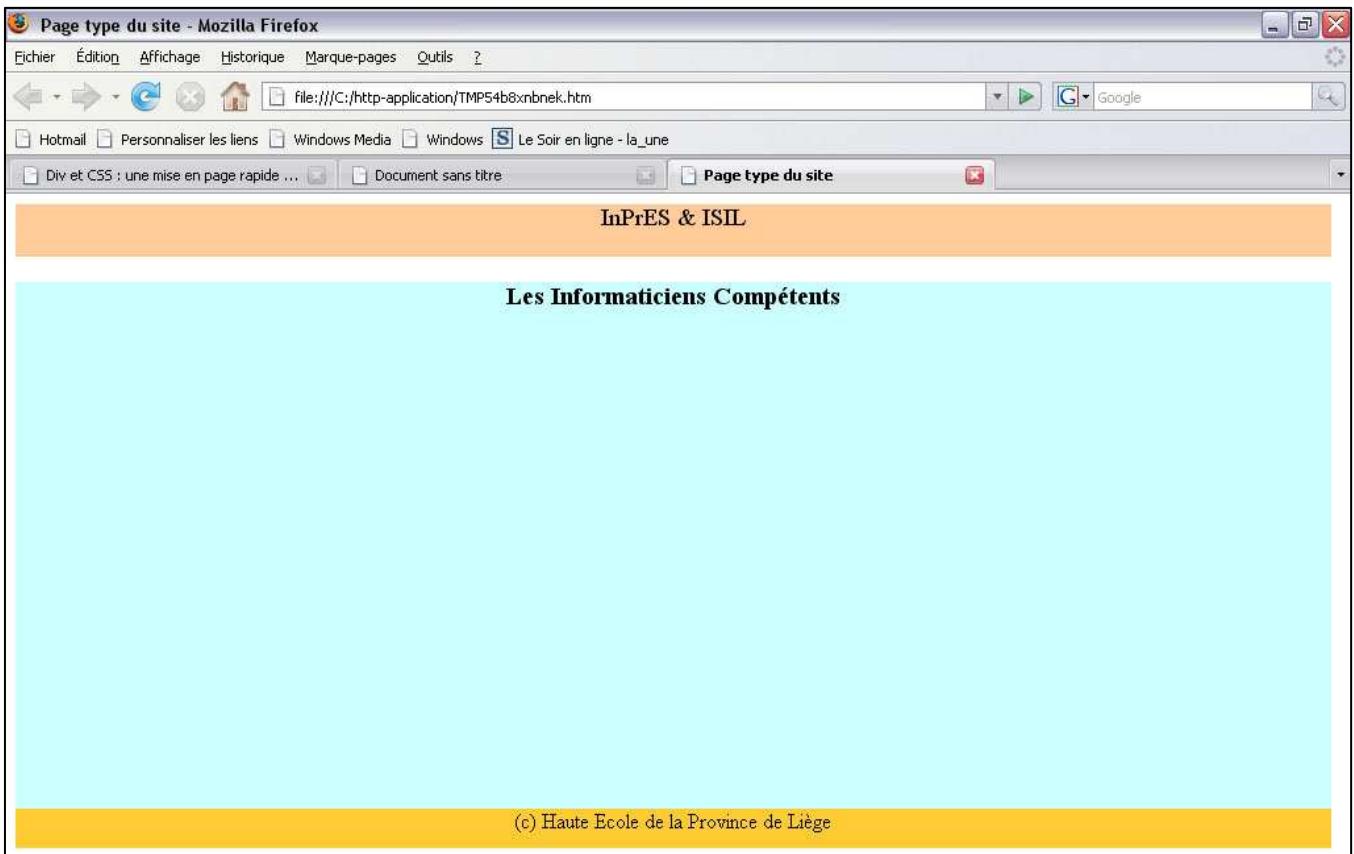
Nous avons ainsi défini trois identifiants pour trois styles au nom bien clair. Utilisons-les dans une page HTML. Cela se fera ainsi :

index.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Page type du site</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link rel="stylesheet" type="text/css" href="structure-page.css">
</head>

<body>
<div id="bandeau">InPrES & ISIL</div>
<div id="contenu"><H3>Les Informaticiens Compétents</div>
<div id="piedpage">(c) Haute Ecole de la Province de Liège</div>
</body>
</html>
```

On obtient :



5.2 Une mise en page à 4 zones

Nous allons à présent ajouter une 4^{ème} zone située à gauche et destinée au menu :

structure-page.css (2)

```
div
{
    text-align:center;
}

div#bandeau
{
    width:1000px;
    height:40px;
    background-color:#FFCC99;
    font-size:large;
}

div#contenu
{
    float:left;
    width:900px;
    height:400px;
    background-color:#CCFFFF;
}
```

```
div#menu
{
    float:left;
    width:100px;
    height:400px;
    background-color:#FF6699;
}

div#piedpage
{
    clear:both;
    width:1000px;
    height:30px;
    background-color:#FFCC33;
}
```

mais il nous faut tenir compte de deux problèmes :

- ♦ la zone de menu doit recouvrir partiellement la zone contenu : par défaut, elles se placeront l'une en-dessous de l'autre, mais on peut corriger cela au moyen de la propriété **float** qui permet de positionner un bloc sur son parent (c'est-à-dire le bloc défini juste avant dans le code HTML).
- ♦ la zone de pied de page doit, par contre, se comporter normalement, doit se placer en dessous des autres et pas en-dessous – la propriété **clear** avec la valeur both permet d'exprimer que le pied de page doit se placer à la fois sous le menu et le contenu (on aurait pu demander cela seulement pour menu ou pour contenu avec les valeurs left et right).

Si la page HTML est écrite ainsi :

```
index.htm (2)
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Page type du site</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link rel="stylesheet" type="text/css" href="structure-page.css">
</head>

<body>
<div id="bandeau"><B>InPrES & ISIL</B></div>
<div id="menu"><i>Menu principal</i></div>
<div id="contenu"><H3>Les Informaticiens Compétents</div>
<div id="piedpage">(c) Haute Ecole de la Province de Liège</div>
</body>
</html>
```

nous obtiendrons :



On constate ainsi que *l'usage des divisions et de CSS constitue une alternative aux frames* ...
Nous y reviendrons avec HTML 5.

6. Les styles de tableaux

Il est également possible de définir des styles spécifiques à l'usage des tableaux. En fait, la panoplie disponible est énorme : elle porte tant sur les bordures que sur les espaces entre cellules, les couleurs, les tracés, etc. Très modestement, illustrons quelques possibilités avec la feuille de style suivante :

e-commerce__home-page pour tableaux.css

```
table { border: thick dotted blue; border-spacing: 30px 10px; }
tr { color:red; font-weight: bold; }
td { border: ridge yellow; }
td i { color:blue; font-weight: bold; }
```

Nous y voyons apparaître les propriétés :

- ◆ **border** : la bordure est elle-même composée de
 - width : thin, medium, thick;
 - style : hidden, dotted, dashed, solid, double, groove, ridge, inset, outset ou none (par défaut);
 - color : comme d'habitude.
- notre tag <TABLE> implique donc des bords en pointillés épais bleus;

- ◆ **border-spacing** : qui définit la distance séparant deux cellules; si deux nombres sont précisés, le premier désigne l'espace horizontal, le deuxième le vertical;
 → notre tag <TABLE> aura donc aussi des cellules plus éloignées horizontalement que verticalement;

Les deux styles suivants, concernant <TR> et <TD>, n'ont rien de particulier. Par contre, le dernier est un exemple de "**sélecteur contextuel**" : en fait, il définit le style <I> dans le cas particulier où il intervient au sein d'une cellule de tableau (<TD> et </TD>). Autrement dit, le tag <I> dans une cellule donne de l'italique bleu gras, tandis qu'à l'extérieur, c'est la définition habituelle de <I> qui s'applique.

Utilisons tout cela dans une page HTML :

e-commerce__home-page_avec_tableau.html (2)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>e-commerce : home-page</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">

<link href="styles/e-commerce__home-page%20pour%20tableaux.css"
      type="text/css" rel="stylesheet" media="screen">
</head>
<body>

<h1>Bonjour &agrave; tous ! :-)</h1>

<h2> Vous voici sur LE site du <i>monde paysan</i> !!! </h2>

<table border="1" cols="3" width="100%">
<tbody>
  <tr>
    <td> Christophe DELANTENNE </td>
    <td><i>ma&icirc;tre </i>des bœufs</td>
    <td>7h30-16h30</td>
  </tr>
  <tr>
    <td>Mounawar BENSIMPATI </td>
    <td><i>directeur </i>financier</td>
    <td>9h-15h30</td>
  </tr>
  <tr>
    <td>Denys UNIXALAISE </td>
    <td><i>ma&icirc;tre </i>des grains&nbsp;</td>
    <td>sur rendez-vous<br> </td>
  </tr>
  <tr>
    <td>Claude DUMOLLUSQUE&nbsp;</td>
    <td><i>ma&icirc;tre </i>des visites&nbsp;</td>
    <td>8h-17h</td>
  </tr>
</tbody>
</table>
```

```
</tr>  
  
</tbody>  
</table>  
  
</body>  
</html>
```

Si, à nouveau, l'éditeur ne semble qu'à moitié intéressé par nos tags remaniés :



le browser se montre beaucoup plus perspicace quant à nos désirs :

N e-commerce : home-page - Netscape

File Edit View Go Bookmarks Tools Window Help

file:///C:/notes-http/notes-http-20 Search

Mail Home Radio My Netscape Search Bookmarks

Bonjour à tous ! :-)

Vous voici sur LE site du *monde paysan* !!!

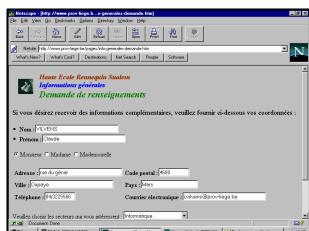
Christophe DELANTENNE	maître des bœufs	7h30-16h30
Moumawar BENSIMPATI	directeur financier	9h-15h30
Denys UNIXALAISE	maître des grains	sur rendez-vous
Claude DUMOLLUSQUE	maître des visites	8h-17h

Document: Done (0.156 secs)



Jusqu'à présent, le rôle du serveur s'est limité à fournir des pages statiques, c'est-à-dire toutes prêtes dans un fichier, éventuellement avec le concours d'une feuille de style. On ne peut pas dire qu'il travaille beaucoup dans de telles circonstances. Cela va changer !

VIII. Les formulaires et les CGIs



La grande histoire véritable est celle des inventions.

(R. Queneau, Bâtons, chiffres et lettres)

1. L'objectif

Une interactivité classique entre un internaute et un serveur est l'entrée d'informations par le client. Ces informations, gérées par le serveur, lui permettent de prendre en compte des demandes du client, de lui transmettre des réponses aux requêtes qu'il a formulées, etc.

L'instrument privilégié pour l'introduction d'informations est le formulaire. Il s'agit bel et bien du formulaire habituel, bien connu des traitements de textes ou des SGBD. Il peut comporter :

- ◆ des zones d'entrées;
- ◆ des boutons radio (une seule possibilité peut être choisie);
- ◆ des cases à cocher (plusieurs possibilités peuvent être retenues);
- ◆ des boutons pousoirs;
- ◆ des boîtes de liste (une ou plusieurs possibilités peuvent être retenues).

L'apparence sous le browser (un vieux !) peut être celle-ci:

Demande d'informations sur la ferme - Netscape

Eichier Edition Afficher Aller Communicator Aide

Précédent Suivant Recharger Accueil Rechercher Guide Imprimer Sécurité Shop Arrêter

Signets Adresse : file:///D|/notes-http/home-farme-demande-info.html Infos connexes

Instant Message Internet Nouveautés Avoir Membres Marché

La ferme "Gédéon et Marie"

Demande d'information

Si vous désirez des informations complémentaires, veuillez fournir ci-dessous vos coordonnées:

Nom : Prénom :

Monsieur Madame Mademoiselle

Exploitant Enseignant Parent Fournisseur

Adresse : Code postal :
 Ville : Pays :
 Téléphone : Courrier électronique :

Pédagogique ▲
 Cultures ▼

Veuillez choisir les secteurs qui vous intéressent :

Document : chargé

Démarrer Microsoft Word Paint Shop ... http://www... Demande d... Demande... 09:55

Il faut remarquer que, pour la première fois, le serveur va jouer un rôle différent de celui d'un simple fournisseur de pages WEB : il devra en effet acquérir les informations transmises, les stocker d'une manière ou d'une autre, éventuellement générer une réponse ... Les choses vont donc se compliquer.

2. Les éléments de base d'un formulaire

Tous les éléments simples d'entrée (zones de textes, boutons, cases à cocher) sont décrites au moyen de la balise <INPUT>. Celle-ci est utilisée habituellement avec les attributs

- ◆ **TYPE**, qui indique de quel type d'élément il s'agit ("text", "radio" ou "checkbox");
- ◆ **NAME**, qui donne un nom à l'élément et qui permettra au serveur d'identifier les données reçues : celui-ci réceptionne en effet les entrées sous la forme d'un couple <nom élément>/<valeur>; ce nom est le même pour tous les boutons radios appartenant à un même groupe;
- ◆ **VALUE**, qui indique une chaîne initiale pour une zone de texte et la valeur correspondante à un bouton radio.

Le texte accompagnant ces éléments est placé comme pour tout autre tag. Dans le cas d'un élément "text", un attribut supplémentaire **SIZE** permet de fixer la longueur de la zone.

Les premières lignes du formulaire évoqué ci-dessus sont générées par le code HTML suivant :

```
<B>Nom :</B><INPUT type="text" name="nom" size=30>

<B>Prénom :</B><INPUT type="text" name="prenom" size=30>

<P><INPUT type="radio" name="intitulé" value="Monsieur"> Monsieur
<INPUT type="radio" name="intitulé" value="Madame"> Madame
<INPUT type="radio" name="intitulé" value="Mademoiselle"> Mademoiselle </P>

<p><input type="checkbox" name="exploitant"> Exploitant
<input type="checkbox" name="enseignant"> Enseignant
<input type="checkbox" name="parent"> Parent
<input type="checkbox" name="fournisseur"> Fournisseur
```

Rien n'interdit de placer des éléments d'entrée dans un tableau. C'est que nous avons fait pour les éléments suivants du formulaire :

```
<table>

<tr>
<td><B>Adresse :</B><INPUT type="text" name="adresse" size=30></td>
<td><B>Code postal :</B><INPUT type="text" name="codePostal" size=10></td>
</tr>

<tr>
<td><B>Ville :</B><INPUT type="text" name="ville" size=30></td>
```

```
<td><B>Pays :</B><INPUT type="text" name="pays" size=20></td>
</tr>

<tr>
<td><B>Tél&phone :</B><INPUT type="text" name="telephone" size=10></td>
<td><B>Courrier électronique :</B><INPUT type="text" name="email" size=30></td>
</tr>

</table>
```

On peut encore signaler que :

- ◆ le type "password" est analogue au type "text", si ce n'est que les caractères tapés sont remplacés à l'écran par des points ou des astérisques; il n'y a en fait aucun encryptage : ce sont les données en clair qui seront envoyées au serveur.
- ◆ l'attribut **CHECKED** permet de désigner le choix par défaut pour les boutons radio et les cases à cocher :

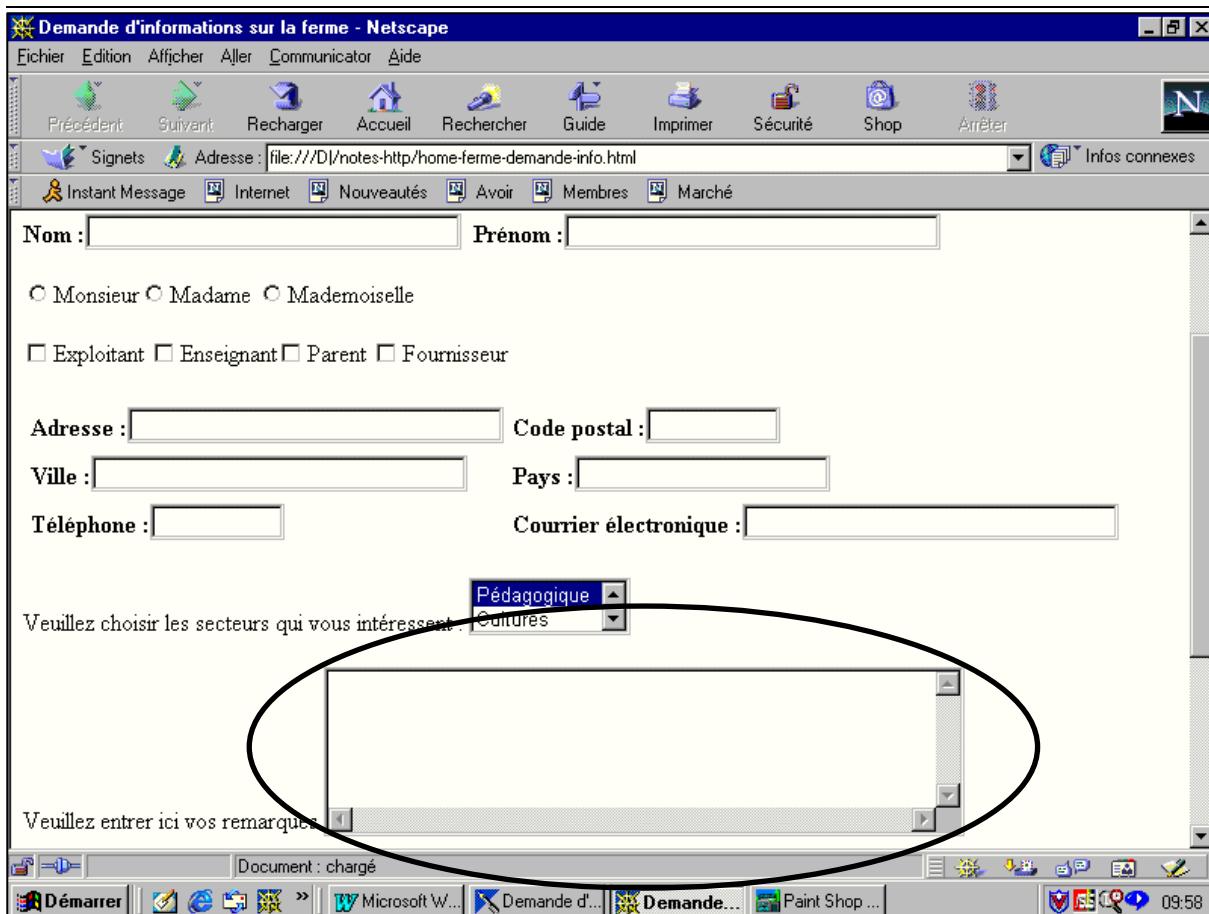
```
<P><INPUT type="radio" name="intitulé" value="Monsieur" CHECKEDCHECKED

```

- ◆ les boutons pousoirs ont le type "button"; mais aucun mécanisme HTML simple n'est prévu pour gérer l'événement correspondant à l'appui du bouton (sauf cas particuliers que nous allons devoir envisager plus loin). Il faut dans ce cas utiliser des CGI ou des scripts.
- ◆ lorsque le texte attendu risque d'être fort long (par exemple, pour demander les réactions d'un utilisateur), on préfère à la balise <INPUT> la balise <TEXTAREA> dont les attributs ROWS et COLS fixent la dimension à l'écran. Pour notre exemple, le code HTML :

```
<P>Veuillez entrer ici vos remarques
<TEXTAREA NAME="commentaires" ROWS=5 COLS=50></TEXTAREA>
```

donne comme effet :



On peut constater la présence de barres de défilement.

3. Les boîtes de listes

Ce contrôle semble plus complexe, mais s'implémente en fait très facilement. Il suffit en effet d'utiliser la balise **<SELECT>** suivie de l'attribut **NAME** qui donne un nom à l'élément. Les différents items de la liste sont à citer chacun derrière un tag **<OPTION>**. Pour notre exemple :

```
<P>Veuillez choisir les secteurs qui vous intéressent :
<SELECT Name="secteur">
<OPTION>P&eacute;dagogique
<OPTION>Cultures
<OPTION>Elevage
</SELECT></P>
```

Quelques raffinements :

- ♦ une valeur par défaut peut être désignée par l'attribut **SELECTED** :

```
<OPTION SELECTED>P&eacute;dagogique
```

- ♦ le nombre d'éléments affichés à l'écran est fixé par l'attribut **SIZE** de la balise **<SELECT>**; un ascenseur apparaît alors si nécessaire;

- ♦ cette même balise admet aussi la balise MULTIPLE pour permettre la sélection de plusieurs items. Cependant, le nom de la boîte de liste reste unique, si bien que c'est la concaténation des diverses valeurs qui forme la valeur associée au nom. Bon courage au serveur ! Peu recommandé ...

4. La mise en place du formulaire

Jusqu'à présent, nous n'avons fait que de mettre en place des éléments d'entrée très divers. Mais rien pour l'instant n'indique qu'ils font partie d'un formulaire à traiter globalement. C'est là le rôle de la balise <**FORM**> qui marque le début du formulaire, avec bien sûr </FORM> pour en indiquer la fin. Pour notre exemple :

```
<P><FORM ...>

<b>Nom:</b><input type="text" name="nom" size=30></b>&nbsp; <b>Prénom</b>
</b><input type="text" name="prenom" size=30>

<p><input type="radio" name="intitulé" value="Monsieur">Monsieur<input type="radio"
name="intitulé" value="Madame">Madame&nbsp;<input type="radio" name="intitulé"
value="Mademoiselle">Mademoiselle

<p><input type="checkbox" name="exploitant">Exploitant&nbsp;<input type="checkbox"
name="enseignant">Enseignant<input type="checkbox" name="parent">Parent&nbsp;<input
type="checkbox" name="fournisseur">Fournisseur
<br>&nbsp;

<table>
<tr>
<td><b>Adresse :</b><input type="text" name="adresse" size=30></td>
<td><b>Code postal :</b><input type="text" name="codePostal" size=10></td>
</tr>
<tr>
<td><b>Ville :</b><input type="text" name="ville" size=30></td>
<td><b>Pays :</b><input type="text" name="pays" size=20></td>
</tr>
<tr>
<td><b>Téléphone :</b><input type="text" name="telephone" size=10></td>
<td><b>Courrier électronique :</b><input type="text" name="email" size=30></td>
</tr>
</table>

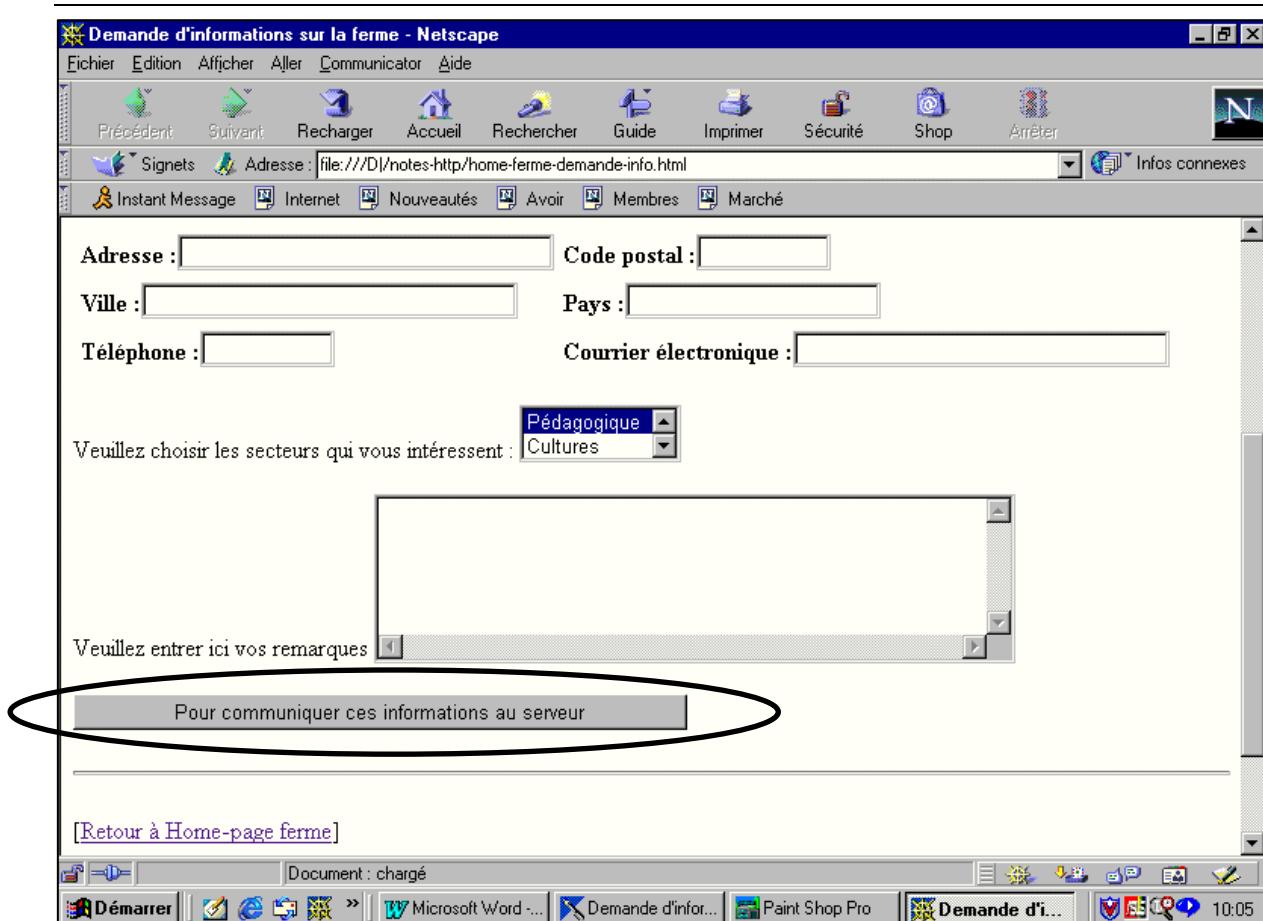
<p>Veuillez choisir les secteurs qui vous intéressent :&nbsp;<select Name="secteur"
SIZE=2><option
SELECTED>Pédagogique<option>Cultures&nbsp;<option>Elevage&nbsp;</select>

<p>Veuillez entrer ici vos remarques&nbsp;<textarea NAME="commentaires" ROWS=5
COLS=50></textarea>

<P><input TYPE="submit" ...></P>

<P></FORM></P>
```

Un formulaire possède le plus souvent un bouton de type ‘SUBMIT’ (mais pas plusieurs). Les informations sont effectivement envoyées au serveur lorsque l'utilisateur clique sur ce bouton qui aura, pour notre exemple, l'aspect suivant :



et qui se déclare par :

```
<INPUT TYPE="submit" Value="Pour communiquer ces informations au serveur">
```

Le tag <FORM> réclame deux attributs non triviaux :

◆ METHOD

Il s'agit ici de déterminer comment les données seront récupérées par le serveur. Avec la méthode **GET**, les données recueillies au moyen d'un formulaire sont placées dans une variable d'environnement appelée typiquement **QUERY_STRING**. Evidemment, il y a intérêt à ce que le nombre de caractères qu'elle peut contenir soit suffisant, sans quoi une troncature des données sera effectuée. Il est donc préférable d'utiliser l'autre méthode, soit **POST** : ici, les données envoyées au serveur servent d'entrée au programme de traitement des données, exactement comme si elles étaient entrées au clavier – il s'agit donc d'une redirection des entrées.

◆ ACTION

Il faut ici donner le nom complet du programme qui, sur le serveur, traitera les données reçues. Ce programme est d'un nature un peu particulière : c'est un **''gateway''**. Si bien que la balise <FORM> a l'aspect suivant :

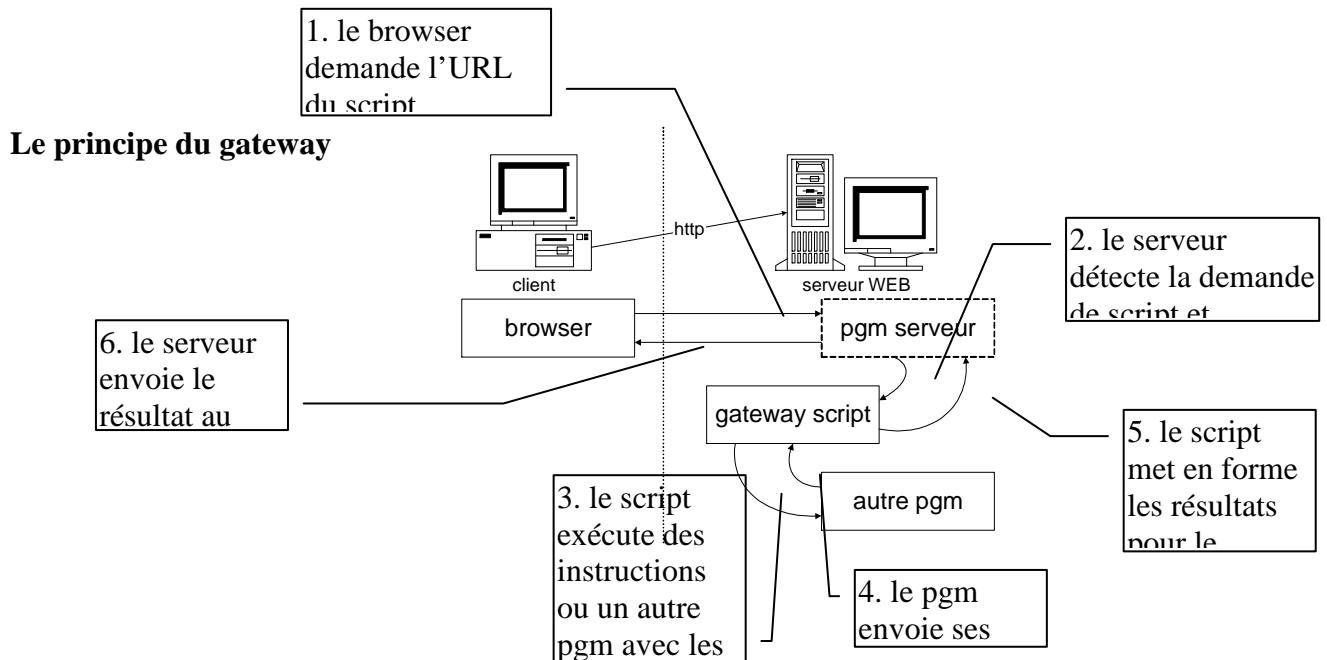
```
<P><FORM METHOD="POST" ACTION="http://www.prov-liege.be/cgi-bin/datain">
```

Mais qu'est-ce qu'un 'gateway' ?

5. Les gateways

Un programme comme celui qui est nécessaire ci-dessus, c'est-à-dire exécuté comme réaction aux informations envoyées par le client, s'appelle un **script de passerelle** ou **gateway script**. Un tel programme pourra aussi renvoyer des informations au serveur, qui les transmettra au client.

Un gateway script est donc un programme qui tourne sur le serveur WEB et qui a été lancé à partir d'une information en provenance d'un interface de navigation.



De tels programmes sont encore appelés des **CGI** (Common Gateway Interface), d'après le nom utilisé dans le monde UNIX. Il peut en fait s'agir de n'importe quel exécutable (un script UNIX, un exécutable résultant de la compilation d'un programme C ou C++ ou encore Cobol ...). Ils sont en général placés dans le répertoire **cgi-bin**, ou celui qui correspond à cette URL.

Pour que de tels CGI fonctionnent, il faut donc impérativement que le serveur soit configuré pour cela. Pour un serveur Apache, il existe une directive ScriptAlias (dans le fichier de configuration /etc/httpd.conf) définissant pour l'alias /cgi-bin/ le répertoire correspondant : toute requête pour une ressource commençant par /cgi-bin/ doit être prise en compte depuis le répertoire associé et doit être traitée en tant que programme CGI. Pour Tomcat, les CGIs sont gérés par une servlet Java, qu'il faut donc installer.

En accord avec le protocole HTTP, chaque script doit d'abord renvoyer un en-tête renseignant le serveur, et donc finalement l'interface de navigation client, sur le type de données créées : s'agit-il d'une page HTML ? ou d'une image ? Cet en-tête n'apparaît jamais à l'utilisateur final, il est seulement interprété. Comme on sait (chapitre I), c'est le rôle du champ de header "Content-type:" suivi du code ad hoc (pour rappel, un tel code est un code **MIME** - Multipurpose Internet Mail Extensions). Toujours pour rappel, voici quelques exemples courants, mais il en existe beaucoup plus :

<i>code</i>	<i>type de données renvoyées</i>
text/plain	texte
text/html	fichier HTML
image/gif	fichier .gif
image/jpeg	fichier .jpeg
application/postscript	fichier postscript
video/mpeg	séquence vidéo MPEG

Le script devra ensuite envoyer des informations en accord avec le type déterminé par l'en-tête. Comme tout client qui envoie une requête à un serveur HTTP précise les types MIME qu'il accepte en retour, puisqu'il fournit dans sa requête des clauses du type :

Accept : text/html

Accept : image/gif

...

c'est au serveur de convertir les données qu'il expédie en retour dans un format que le client est à même d'accepter.

6. Un exemple de gateway

Avant d'entrer dans le traitement d'un formulaire, expérimentons un modeste script CGI. Il a pour tâche de créer une page HTML qui affichera l'heure et la date du serveur. Cette page n'est pas créée dans un éditeur HTML : c'est le programme lui-même qui enverra au client, par l'intermédiaire du serveur, les éléments constituant la page. Etant créée au moment même de la requête, une telle page peut donc être **dynamique**.

Supposons que l'on accédera à cette page en demandant 'HORLOGE'. Ce fichier est en fait le programme CGI, donc un exécutable. Son rôle est de fournir une page HTML qui pourra être interprétée par le browser. Donc, il lui faut :

- ◆ envoyer un en-tête MIME renseignant sur le type de données qui est envoyé;
- ◆ fournir le début de la page (en-tête, premières lignes);
- ◆ afficher l'heure, la date;
- ◆ fournir la fin de la page.

Le programme en question s'écrit en C de la manière donnée ci-dessous. Le seul point délicat est de bien réaliser que toutes les sorties de ce programme (donc, tout ce qui fait l'objet d'instructions comme printf) seront envoyées telles quelles au client par l'intermédiaire du serveur :

HORLOGE.C

```

/* horloge.c */
/* Claude Vilvens */

#include <stdio.h>
#include <time.h>

#define HTML_HEAD "horloge.head"
#define HTML_FOOT "horloge.foot"

#define N_MAX_CAR 1024

FILE * checkFile(char * nFile);
void includeFile (char *nFile);
void chercheTemps ();

int main()
{
    printf ("Content-type: text/html\n\n"); // en-tête HTTP-MIME
    includeFile(HTML_HEAD);           // début de la page
    chercheTemps();                  // heure, date
    includeFile(HTML_FOOT);          // fin de la page
}

FILE * checkFile(char * nFile)
{
    static FILE * fichier;

    fichier = fopen(nFile, "r+");
    if (fichier == NULL)
    {
        fprintf(stderr, "VVV fichier %s non trouve !", nFile); exit(1);
    }
    return (FILE *)fichier;
}

void includeFile (char *nFile)
{
    FILE * fichier = checkFile(nFile);
    char ligne[N_MAX_CAR];

    ligne[0]='\0';
    while (fscanf (fichier, "%s", ligne) != EOF)
    {
        fgetc(fichier); printf("%s\n", ligne); ligne[0] = '\0';
    }
    fclose(fichier);
}

```

```
void chercheTemps ()  
{  
    time_t t;  
    struct tm* pt;  
    char chaine1[80],chaine2[80],chaine3[80];  
  
    t = time(0); pt = gmtime(&t);  
    sprintf(chaine1,"Nous sommes le <h3>%d/%d/%d</h3>\n", pt->tm_mday, pt->tm_mon + 1,  
            pt->tm_year);  
    printf(chaine1);  
    sprintf(chaine2,"<P>Pour les horodateurs : %d \n", pt->tm_yday + 1);  
    printf(chaine2);  
    sprintf(chaine3,"<P>Il est <H3>%d h %d m %d s</H3>\n",  
            pt->tm_hour + 1, pt->tm_min, pt->tm_sec);  
    printf(chaine3);  
}
```

Les fichiers qui sont lus par ce programme forment le début et la fin de la page :

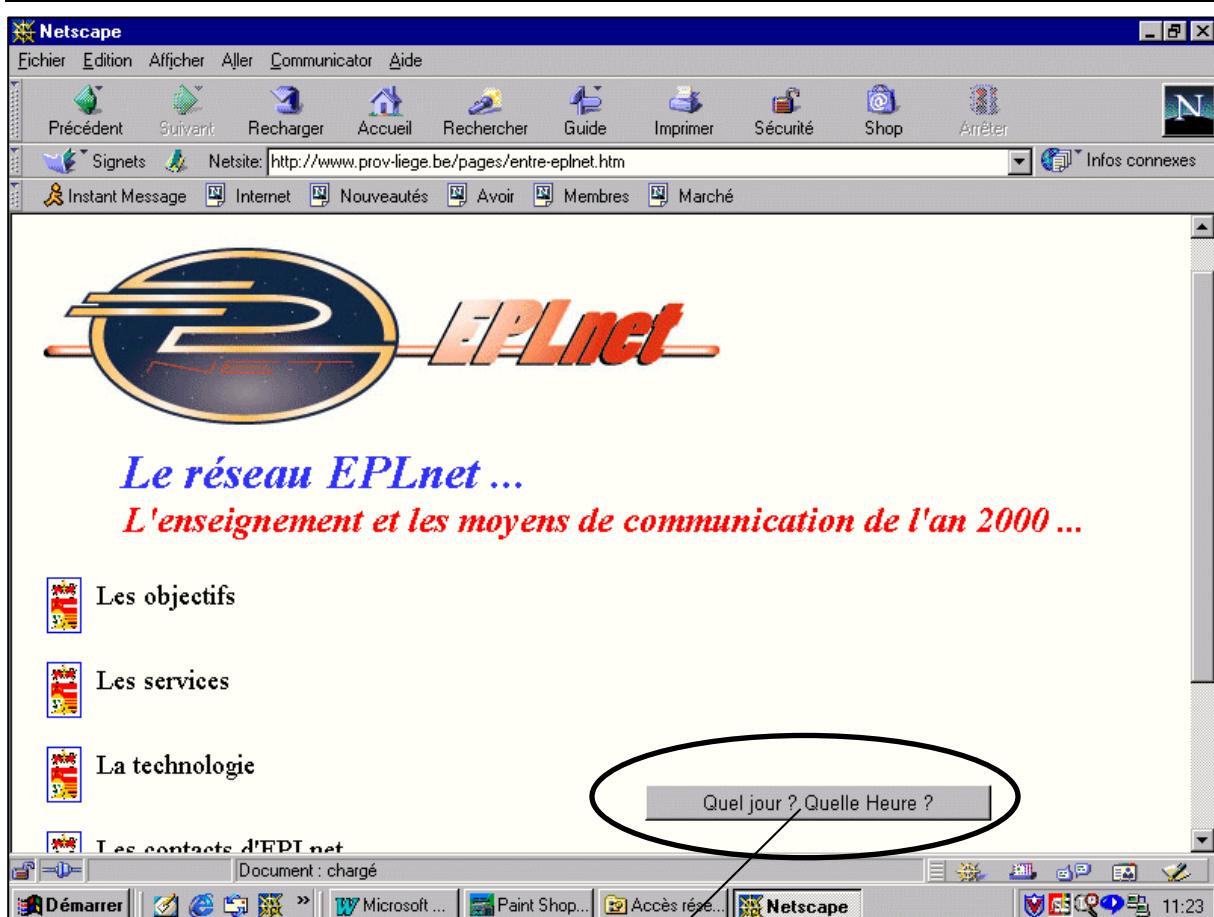
horloge.head

```
<HTML>  
<HEAD>  
</HEAD>  
<BODY>  
<IMAGE src="../images/eplnet.gif" height=83 width=174 border=2>  
<P><H2>Pour le serveur de l'Enseignement de la Province de Liège  
</H2>
```

horloge.foot

```
</H3></H2>  
<P>[<A HREF="../home-eplnet.htm">Retour à la home page EPLnet</A>]</P>  
</body>  
</HTML>
```

La compilation du programme C sur le serveur donne un exécutable qu'il suffit, pour notre exemple, de baptiser ‘HORLOGE.HTML’. Et le tour est joué ...



Pour le serveur de l'Enseignement de la Province de Liège

Nous sommes le

9/1/00

Pour les horodateurs : 9

Il est

13 h 12 m 35 s

généré par le programme C

7. Le CGI d'un formulaire

Revenons à présent à notre formulaire. Lorsque l'on appuie sur le bouton SUBMIT, le serveur devra lancer un programme CGI qui recevra en entrée une chaîne de caractère du type :

nomDonnée=valeur&nomDonnée=valeur&...

où "nomDonnée" est le nom donné par l'attribut NAME de chaque élément du formulaire. On peut donc constater que

- ◆ le caractère ‘&’ sépare un couple <nom de donnée><valeur de la donnée>;
- ◆ le caractère ‘=’ sépare le nom d’une donnée de sa valeur.

On conçoit donc sans peine que le principal travail du CGI sera de décomposer la chaîne reçue de manière à isoler les données pour pouvoir les enregistrer ou les rechercher dans un fichier ou une table de base de données. Ici, il se contentera de les afficher dans la page renvoyée au client (au moyen de printf()).

Avant de fournir le programme C de base, on peut encore signaler que :

- ◆ un certain nombre de **variables d'environnement** contiennent des renseignements concernant la connexion avec le client (REQUEST_METHOD contient le nom de la méthode utilisée (POST ou GET), CONTENT_TYPE contient le type de données envoyées, CONTENT_LENGTH contient le nombre de bytes reçus par le serveur, etc. On récupère leur contenu au moyen de la fonction C **getenv()**.
- ◆ dans la chaîne envoyée par le client,
 - ⇒ un blanc est remplacé par ‘+’;
 - ⇒ les caractères spéciaux, comme ‘\$’ ou les caractères accentués, sont automatiquement remplacés par le browser en leur forme hexadécimale, soit “%XX”.

Il convient évidemment que le CGI remplace ces caractères par leur signification originale.

FORMULA.C

```
/* FORMULA.C */
/* Claude Vilvens */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NB_BYTES 1000
#define N_MAX_CAR 1024
/* pour lire les fichiers head et foot */

void traiteChaine (char *s);
void includeFile (char *nFile);

int main()
{
    int nbBytes, nbData, nbBytesEgalite, nbBytesName, i;
    char    ** dataName, /* le nom des donnees */
            ** data,       /* les donnees */
            bytesRecus[MAX_NB_BYTES+1], /* pour recevoir les donnees */
            * methode = getenv("REQUEST_METHOD"),
            * typeData = getenv("CONTENT_TYPE"),
            * snbBytes = getenv("CONTENT_LENGTH"),
            * sep;
```

```
int rc;
char pasDeData=1;

printf ("Content-type: text/html\n\n");
includeFile('formula.head');

printf("(Methode utilisee = %s\n",methode);
printf("Type des donnees = %s\n",typeData);

/* combien de bytes envoyes par le client */
nbBytes = atoi(snbBytes);
if (nbBytes==0)
{
    printf ("Aucune information recue !");
    return 1;
}
else printf("Le serveur a recu : %d bytes)<p> ", nbBytes);

rc = fread(bytesRecus,1,nbBytes, stdin);
if (!rc)
{
    printf("Erreur de lecture des donnees\n");
    return 2;
}
bytesRecus[nbBytes]='\0';

/* combien de donnees ? */
nbData = 0;
sep = strchr(bytesRecus, '&');
while (sep)
{
    nbData++;
    if (sep+1 != 0) sep = strchr(sep+1, '&');
}

printf("Le serveur a reconnu %d data<P>",nbData);
/* allocations */
dataName = (char **)malloc(sizeof(char *)*nbData);
data = (char **)malloc(sizeof(char *)*nbData);

sep = strtok(bytesRecus,"&");
/* sep pointe la sous-chaine limitee par & */
for (i=0; sep; i++)
{
    nbBytesEgalite = strlen(sep);
    nbBytesName = strcspn(sep,"=");
    /* nbre de caracteres avant */
    dataName[i] = (char *)malloc(sizeof(char)*(nbBytesName+1));
    strcpy(dataName[i], sep);
    dataName[i][nbBytesName] = 0;
```

```
        traiteChaine(dataName[i]); printf("%s = ",dataName[i]);

        if (nbBytesEgalite > nbBytesName)
        {
            data[i] = (char *)malloc(sizeof(char)*
                (nbBytesEgalite - nbBytesName));
            strcpy(data[i], sep+nbBytesName+1);
            traiteChaine(data[i]);
            if (strcmp(data[i],"")!=0)
            {
                printf("<FONT color=#0000FF>%s</font><BR>",data[i]);
                /* instructions d'écriture dans un fichier .... */
                if (pasDeData && i!= 8) pasDeData=0;
            }
            else
            {
                printf("?<BR>");
            }
        }
        else printf("?<BR>");
        sep = strtok(0,"&");
    }

    if (nbData>0)
    {
        int i;
        for (i=0; i<nbData; i++)
        {
            if (data[i])
                free(data[i]);
            if (dataName[i])
                free(dataName[i]);
        }
        if (data)
            free (data);
        if (dataName)
            free(dataName);
    }

    if (pasDeData)
        puts("<p><STRONG>Aucune donnee utile enregistree !</STRONG>");  

    else puts("<P><STRONG><BLINK>Traitement terminé !</BLINK></STRONG>");

    includeFile('formula.foot');

    return 0;
}
```

```

void traiteChaine (char *s)
/* Epure une chaîne reçue du client:
- les + deviennent des espaces
- les caractères accentués, qui génèrent des codes hexa, sont restitués */
{
    int ia, ic;      /* i pour avancer, i pour changer */
    char hex[4];    /* pour les codes %xx */

    for (ia=0, ic=0; s[ia] ; ia++, ic++)
    {
        if (s[ic]=='+') s[ic]=' ';
        else if (s[ic]=='%')
        {
            hex[0]=s[ia+1]; hex[1]=s[ia+2];
            hex[2]=0;
            s[ic]=strtol(hex, 0, 16);
            /* conversion d'un hexa en un long */
            ia += 2;
        }
        else s[ic]=s[ia];
    }
    s[ic]=0;
}

FILE * checkFile(char * nFile)
{
    ...
}

void includeFile (char *nFile)
{
    ...
}

```



Après le côté serveur, le côté client. On ne peut pas dire qu'il travaille beaucoup pour l'instant, puisqu'il se contente de prendre des requêtes et de refléter les réponses du serveur. Cela va changer !

IX. Une petite introduction à JavaScript



L'humour est une façon de remettre en question les choses qu'on considère comme intouchables.

[Jean-Louis Bory, Entretien avec Bernard Pivot]

1. Un langage de script

JavaScript est un langage de "scripting" utilisé dans le développement Web orienté client, initialement développé vers 1995 par Netscape sous le nom de LiveScript. Son créateur est un certain Brendan Eich, informaticien américain en activités chez Netscape puis dans le projet Mozilla, qui conduira à la fondation Mozilla puis à Mozilla Corporation dont il est devenu un directeur..

Le langage de Eich fut adopté par Sun en 1995 et fut alors rebaptisé en "JavaScript" (tiens donc ;- !). A priori, il était destiné aux seuls browsers de Netscape, mais Microsoft Internet Explorer s'y est très vite intéressé. Plus récemment, Mozilla Firefox a également pris en charge le célèbre langage. Car célèbre, il l'est devenu assez vite, permettant aux développeurs de site Web non programmeurs chevronnés de néanmoins enrichir l'interactivité de leurs sites.

JavaScript permet donc d'ajouter à du code HTML classique des scripts (c'est-à-dire une suite d'instructions) réalisant des opérations spécifiques que le simple HTML n'est pas capable de réaliser : aide et contrôle sur les entrées, affichages de GUIs "intelligents", animations, etc.

Ces scripts seront interprétés et exécutés par le user-agent du **client** (donc, le browser) sans intervention du serveur HTTP : c'est évidemment la grosse différence avec les CGIs, servlets/JSP Java, PHP, Perl et ASP/.NET qui sont des technologies **serveur**.

Ces scripts sont libellés "en clair" dans la page HTML qui leur sert d'hôte (il s'agit bien d'une **interprétation**). C'est évidemment la différence avec les applets Java, qui parviennent au client sous forme de bytecode (qui résulte d'une forme particulière de **compilation**) et non pas sous la forme d'un source Java compréhensible. L'existence de ce code JavaScript clair est évidemment une faiblesse par rapport aux attaques des hackers – et cela lui a été suffisamment reproché.

Au chapitre des reproches, on cite aussi souvent les limites du langage (si on le compare à Java – évidemment, la barre est haute ;-)) et son côté faussement "orienté objet" jouant sur l'ambiguité relative du terme (en fait, il manque un certain nombre de piliers de l'OO, comme l'encapsulation, l'héritage, le polymorphisme, les exceptions, etc). La syntaxe générale est tout de même fort proche de celle de Java.

Tout comme lui a été souvent reproché les problèmes de compatibilité apparaissant en passant d'un browser à un autre : on ne peut le nier, mais de là à en faire un argument de rejet définitif ... De plus, l'émergence d'AJAX a évidemment suscité un nouvel engouement pour JavaScript, tout comme les nouvelles plates-formes Java ont relancé l'intérêt pour les scripts en général (Ruby).

Ce chapitre introductif s'adresse à des développeurs Java ou C#, qui possèdent donc déjà de solides connaissances en POO. L'objectif est de souligner les analogies et les différences de JavaScript par rapport à ces deux langages OO professionnels, que ce modeste langage de script ne peut en aucun cas penser à détrôner ...

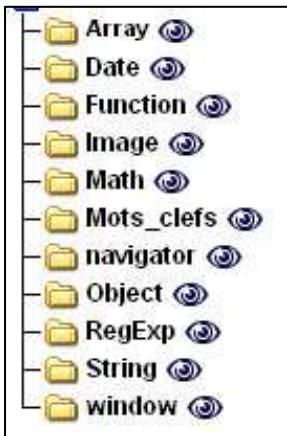
2. Les "objets" implicites de JavaScript

Si l'on considère une page HTML classique contenant un formulaire, le browser et JavaScript y distinguent divers "objets" :

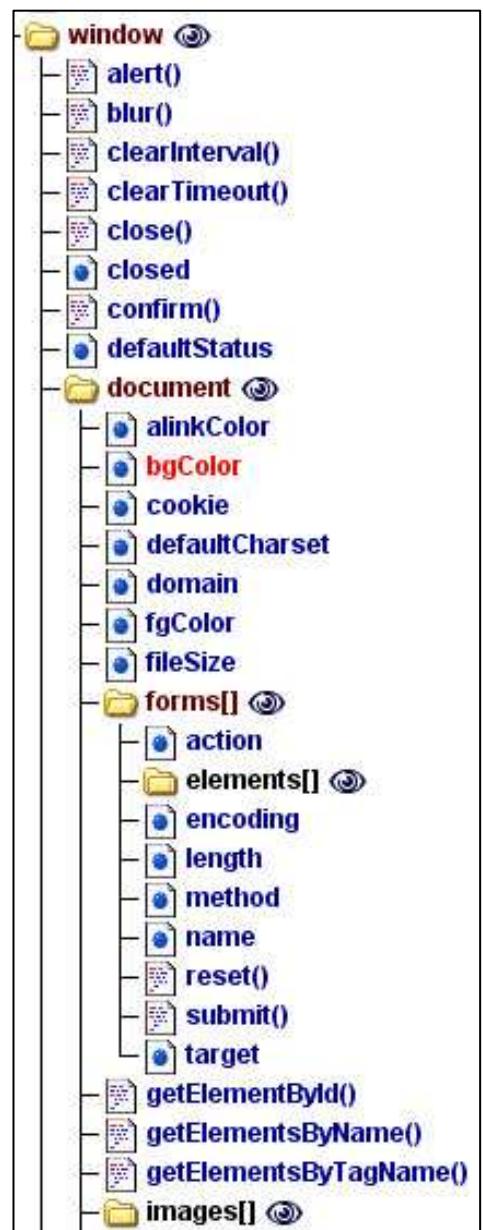
- ◆ la fenêtre, au sens du système d'exploitation – l'objet nommé **window** sert à la représenter;
- ◆ la page HTML que cette fenêtre permet de visualiser grâce à la traduction réalisée par le browser : l'objet **document** la représente;
- ◆ si la page contient des contrôles (comme des boutons, des zones d'entrée, ...), ceux-ci constituent alors un formulaire que l'objet **form** représente.

Ces trois objets ne se situent pas sur le même pied : window contient document qui contient form – autrement dit, on pourrait dire que document est un membre de window, et que form est un membre de document. Mais, en fait, window peut être ignoré car il est sous-entendu par défaut et document peut se manipuler lui aussi directement – mouais, vachement rigoureux ;-) ... En réalité, on peut distinguer :

- ◆ un certain nombre de "classes" et d'"objets" globaux, dans le sens qu'ils ne sont pas intégrés dans une autre classe ou un autre objet :



- ◆ les objets "membres" des précédents, eux-mêmes susceptibles de contenir d'autres membres :



Que contiennent ces objets ? Un certain nombre de méthodes mais aussi un certain nombre de propriétés au sens des Java Beans ou de C#/.NET. Citons en brièvement quelques-unes.

3. L'objet window

C'est l'objet de référence quand le programmeur n'en spécifie pas. Autrement dit, écrire

window.alert()

ou

alert()

signifie la même chose : l'appel de la méthode alert() de l'objet window. Les possibilités les plus utiles de celui-ci sont :

window	
■ name	nom de la fenêtre
■ defaultStatus ■ status	texte par défaut de la barre d'état, texte en cas d'événement
● resizeTo(largeur, hauteur)	taille de la fenêtre
● focus()	acquisition du focus
● alert()	boîte de dialogue d'information – bouton Ok
● confirm()	boîte de dialogue de confirmation – boutons Ok et Cancel
● prompt()	boîte de dialogue d'introduction d'un texte avec bouton Ok
■ document	la page-document contenu dans la fenêtre

4. L'objet document et son contenu

Il représente en fait la page HTML proprement dite et permet d'accéder à tout ce qu'il contient. En abrégé :

document	
■ bgColor ■ fgColor	couleur de fond et de tracé
■ URL	URL de la page en cours
■ mimeType	type mime de la page
■ title	titre de la page
● write()	écrit dans la page à la position de l'appel du script
■ form[]	tableau permettant l'accès aux différents formulaires se trouvant dans la page (en passant le nom du formulaire ou sa position numérotée à partir de 0) – sa variable membre length permet de savoir combien il y en a.

forms[]	
<input type="checkbox"/> action	les éléments correspondant de la balise <FORM>
<input type="checkbox"/> methode	envoyer le formulaire au serveur
<input type="checkbox"/> name	réinitialiser les champs du formulaire
<input type="checkbox"/> elements[]	tableau permettant l'accès aux différents éléments du formulaire visé se trouvant dans la page (mais on peut aussi obtenir un élément par son nom)

En ce qui concerne la navigation au sein des composants d'une page contenant des formulaires, il est souvent plus simple d'utiliser le mot réservé **this** qui désigne l'objet courant (c'est-à-dire celui dans lequel le code que l'on écrit se trouve).

On peut trouver une référence complète des classes, objets, méthodes et propriétés sur <http://www.toutjavascript.com/reference/reference.php> (dont proviennent les deux copies-écran ci-dessus).

5. Un script basique

Pour débuter, nous allons nous contenter de placer quelques instructions JavaScript dans une page HTML. Ces instructions doivent se trouver entre des balises `<script></script>`, plus précisément :

zone JavaScript

```
<script language="JavaScript" >
<!--
...
//-->
</script>
```

La zone de commentaires HTML `<!-- ... -->` n'a pour rien d'autre pour objectif que de masquer le script pour les browsers incapables d'utiliser JavaScript. On peut alors rédiger le script dans la zone ainsi déterminée en tenant compte de ce que :

- ◆ Java Script est "case sensitive";
- ◆ il utilise le code ASCII 7 bits (des caractères accentués comme "é" ou "à" ne peuvent apparaître que dans les chaînes de caractères constantes);
- ◆ si l'on veut désigner les caractères " et ', il faut évidemment utiliser les échappements \" et \';
- ◆ le browser lit la page séquentiellement et doit donc disposer de tous les éléments nécessaires lorsqu'il traite chaque instruction.

accueil-js0.htm

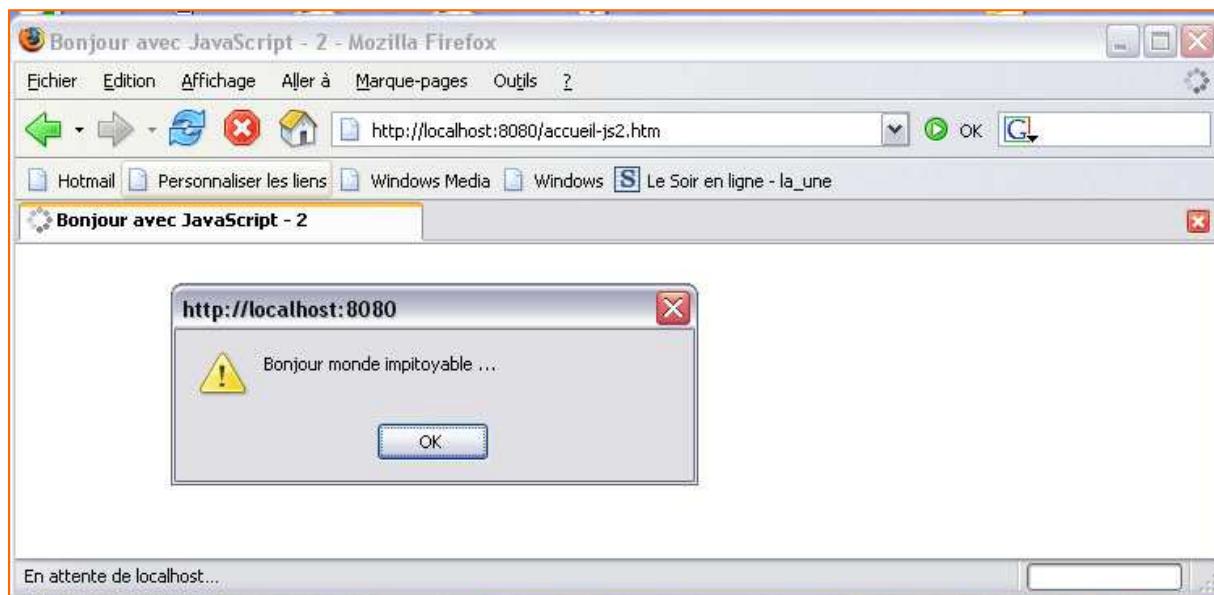
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Bonjour avec JavaScript</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

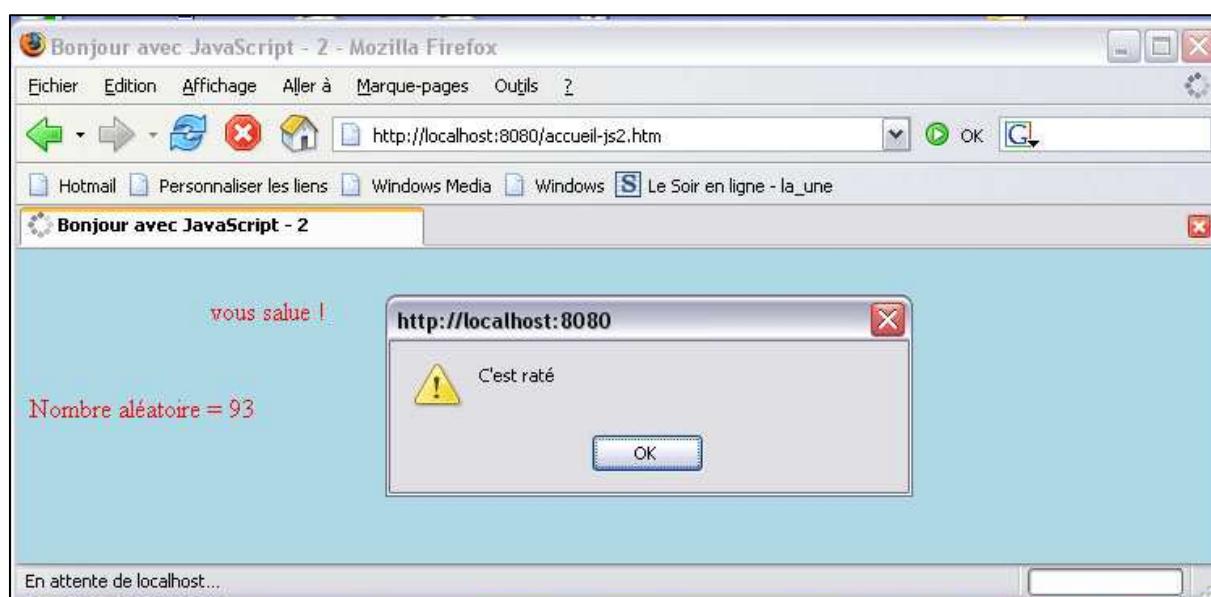
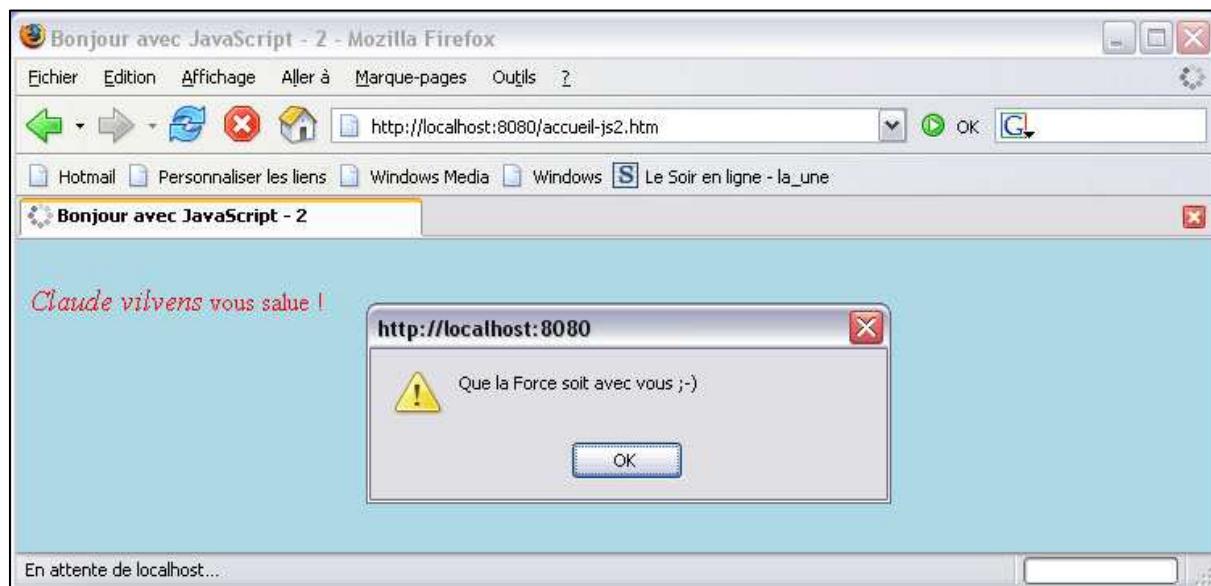
<body>
<h3>Bienvenue :-)</h3>

<script language="JavaScript" >
<!--
window.alert("Bonjour monde impitoyable ...");
document.bgColor="lightBlue";
nom="Claude vilvens"; // variable globale
document.write(nom.big().blink().italics()+" vous salue ! <br>");
var msg="Que la Force soit avec vous ;-)"; // variable locale si utilisée dans une fonction
alert(msg);
document_fgColor="red";
x=Math.round(Math.random()*100);
document.write("Nombre aléatoire = "+x);
pair=x%2==0;
if (pair) alert ("You win");
else alert ("You lost");
//-->
</script>

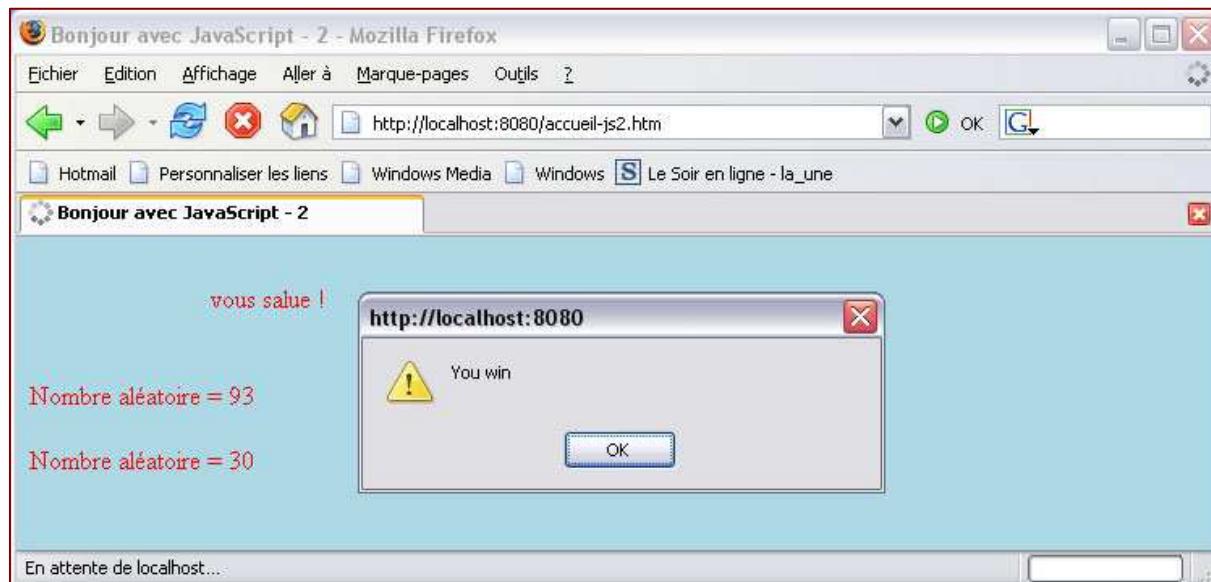
</body>
</html>
```

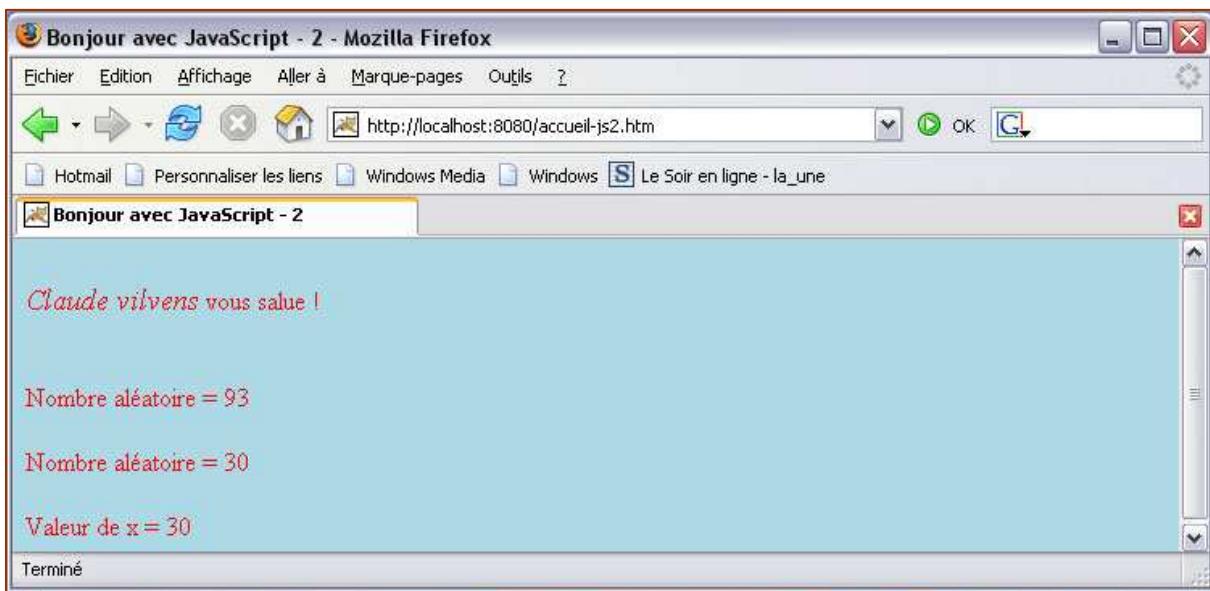
Le résultat est assez prévisible :





ou encore :





Le résultat est identique avec Internet Explorer. A remarquer que Firefox fournit une console JavaScript :



qui est fort utile pour corriger les erreurs de scripting.

6. Les mots réservés de JavaScript

Avant de poursuivre, il faut savoir qu'un certain nombre d'identificateurs sont réservés par le langage, même s'il n'en fait parfois encore rien :-o En fait, ce sont des mots issus de Java le plus souvent, parfois de C++ ou même de Pascal. Ils sont repris dans le tableau ci-dessous (les mots effectivement utilisés sont en gras) :

abstract	boolean	break	byte	case
catch	char	class	const	continue
default	delete	do	double	else
export	extends	false	final	finally
float	for	function	goto	if
implements	import	in	instanceof	int
long	native	new	null	package
private	protected	public	return	short
static	super	switch	synchronized	this
throw	throws	transient	true	try
typeof	var	void	while	with

7. Les scripts externes

Il est possible de mémoriser les scripts dans des fichiers distincts, à l'extension js. On y gagne évidemment en clarté, bien que la compréhension de la page s'en trouve parfois affectée. Pour l'exemple précédent, on peut imaginer le fichier suivant :

basique.js

```
// JavaScript Document
<!--
window.alert("Bonjour monde vraiment impitoyable ...");
document.bgColor="lightBlue";
nom="Claude vilvens"; // variable globale
document.write(nom.big().blink().italics()+" vous salue ! <br>");
var msg="Que la Force soit avec nous ;-); // variable locale si utilisée dans une fonction
alert(msg);
document_fgColor="red";
x=Math.round(Math.random()*100);
document.write("Nombre aléatoire = "+x);
pair=x%2==0;
if (pair) alert ("You win");
else alert ("You lost");
//-->
```

tandis que la page HTML y fait référence :

accueil-js1.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Bonjour avec JavaScript en fichier externe</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body>
<h3>Bienvenue :-)</h3>

<script language="JavaScript" src="basique.js"></script>

</body>
</html>
```

Bien sûr, ceci ne change rien à l'exécution :



Du point de vue du trafic réseau, les choses se passent comme pour une applet Java.
Autrement dit, une première requête HTTP réclame la page HTML et une deuxième demande le fichier JavaScript dont l'utilisation a été détectée. On voit très bien cela au moyen d'un sniffer (ici, Ethereal) :

1 0.000000 IntelCor_60:82:f2 Broadcast ARP who has 192.168.1.8? Tell 192.168.1.2
2 0.000013 Micro-St_71:02:09 IntelCor_60:82:f2 ARP 192.168.1.8 is at 00:0c:76:71:02:09
3 0.001815 192.168.1.2 192.168.1.8 TCP 2014 > 8080 [SYN] Seq=0 Len=0 MSS=1460
4 0.001847 192.168.1.8 192.168.1.2 TCP 8080 > 2014 [SYN, ACK] Seq=0 Ack=1 win=65535 Len=0 MSS=1460
5 0.003130 192.168.1.2 192.168.1.8 TCP 2014 > 8080 [ACK] Seq=1 Ack=1 win=17520 Len=0
6 0.029007 192.168.1.2 192.168.1.8 HTTP GET /accueil-js1.htm HTTP/1.1
7 0.033602 192.168.1.8 192.168.1.2 HTTP HTTP/1.1 304 Non Modifi\x351
8 0.050370 192.168.1.2 192.168.1.8 HTTP GET /basique.js HTTP/1.1
9 0.055764 192.168.1.8 192.168.1.2 HTTP HTTP/1.1 304 Non Modifi\x351
10 0.267282 192.168.1.2 192.168.1.8 TCP 2014 > 8080 [ACK] Seq=976 Ack=185 win=17336 Len=0
11 5.588892 192.168.1.2 192.168.1.8 TCP 2014 > 8080 [FIN, ACK] seq=976 Ack=185 win=17336 Len=0
12 5.588933 192.168.1.8 192.168.1.2 TCP 8080 > 2014 [ACK] Seq=185 Ack=977 win=64560 Len=0
13 5.589070 192.168.1.8 192.168.1.2 TCP 8080 > 2014 [FIN, ACK] seq=185 Ack=977 win=64560 Len=0
+ Frame 6 (567 bytes on wire, 567 bytes captured)
+ Ethernet II, Src: IntelCor_60:82:f2 (00:13:ce:60:82:f2), Dst: Micro-St_71:02:09 (00:0c:76:71:02:09)
+ Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 192.168.1.8 (192.168.1.8)
+ Transmission Control Protocol, Src Port: 2014 (2014), Dst Port: 8080 (8080), Seq: 1, Ack: 1, Len: 513
Hypertext Transfer Protocol
+ GET /accueil-js1.htm HTTP/1.1\r\n
Host: 192.168.1.8:8080\r\n
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.12) Gecko/20070508 Firefox/1.5.0.12\r\n
Accept: text/xml,application/xml,application/xhtml+xml+xml, text/html;q=0.9, text/plain;q=0.8, image/png,*/*;q=0.5\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
If-Modified-Since: Sun, 01 Jul 2007 10:37:38 GMT\r\n
If-None-Match: w/"337-1183286258343"\r\n
\r\n
5 0.003130 192.168.1.2 192.168.1.8 TCP 2014 > 8080 [ACK] Seq=1 Ack=1 win=17320 Len=0
6 0.029007 192.168.1.2 192.168.1.8 HTTP GET /accueil-js1.htm HTTP/1.1
7 0.033602 192.168.1.8 192.168.1.2 HTTP HTTP/1.1 304 Non Modifi\x351
8 0.050370 192.168.1.2 192.168.1.8 HTTP GET /basique.js HTTP/1.1
9 0.055764 192.168.1.8 192.168.1.2 HTTP HTTP/1.1 304 Non Modifi\x351
10 0.267282 192.168.1.2 192.168.1.8 TCP 2014 > 8080 [ACK] Seq=976 Ack=185 win=17336 Len=0
11 5.588892 192.168.1.2 192.168.1.8 TCP 2014 > 8080 [FIN, ACK] seq=976 Ack=185 win=17336 Len=0
12 5.588933 192.168.1.8 192.168.1.2 TCP 8080 > 2014 [ACK] Seq=185 Ack=977 win=64560 Len=0
13 5.589070 192.168.1.8 192.168.1.2 TCP 8080 > 2014 [FIN, ACK] seq=185 Ack=977 win=64560 Len=0
+ Frame 8 (516 bytes on wire, 516 bytes captured)
+ Ethernet II, Src: IntelCor_60:82:f2 (00:13:ce:60:82:f2), Dst: Micro-St_71:02:09 (00:0c:76:71:02:09)
+ Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 192.168.1.8 (192.168.1.8)
+ Transmission Control Protocol, Src Port: 2014 (2014), Dst Port: 8080 (8080), Seq: 514, Ack: 93, Len: 462
Hypertext Transfer Protocol
+ GET /basique.js HTTP/1.1\r\n
Host: 192.168.1.8:8080\r\n
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.12) Gecko/20070508 Firefox/1.5.0.12\r\n
Accept: */*\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
Referer: http://192.168.1.8:8080/accueil-js1.htm\r\n
If-Modified-Since: Sun, 01 Jul 2007 10:38:56 GMT\r\n
If-None-Match: w/"515-1183286336687"\r\n
\r\n

8. Le traitement des événements

Par rapport au HTML classique, JavaScript permet d'ajouter le traitement des événements pouvant survenir sur une page. Ces événements concernant le chargement de la page, les déplacements et clicks de la souris, la modification ou la sélection d'un champ de formulaire, l'envoi d'un formulaire, etc. La mise en place d'un gestionnaire d'événement est assez simple :

- ◆ on définit une fonction JavaScript qui contiendra les actions à effectuer si l'événement survient (c'est donc une fonction "callback" ou "listener" ;-));
- ◆ on complète le tag concernant l'élément HTML concerné d'un attribut ayant la forme

`onXXX="fonction()"`

où "onXXX" est l'un des attributs suivants :

événement	signification	balises compatibles
onAbort	chargement interrompu	
onFocus / onBlur	prise / perte du focus	champs de saisie (<input>, <checkbox>, <radiobutton>, <text>, <textarea>), <body>, <frame>
onClick	clic sur un objet	<input>, <checkbox>, <radiobutton>, <text>, <textarea>, <a>
onChange	changement de champ de saisie	<input>
onDoubleClick	double clic	<a>,
onKeyPress / onKeyUp / onKeyDown	touche appuyé, relâchée, maintenue enfoncee	<a>,
onLoad	chargement	<body>, <frame>,
onMouseDown / onMouseUp	un bouton de la souris est enfonce / relache	<a>,
onMouseOver / onMouseMove / onMouseOut	la souris entre sur, dans ou vient de quitter une zone	<a>,
onSubmit / onReset	envoie / annulation des donnees saisies	<form>
onSelect	sélection d'un champ de texte	éléments de <form>

Donc, par exemple :

evt-onclick.htm
<pre><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"> <html> <head> <title>Fais-moi mal ;-)</title> <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"></pre>

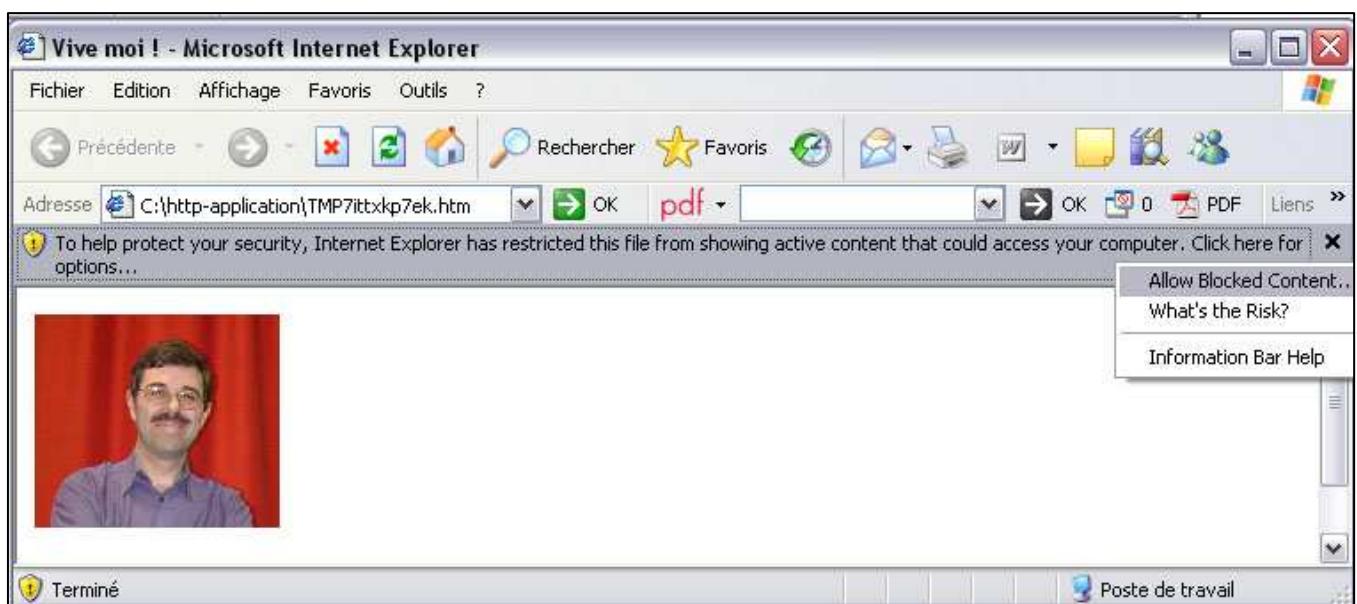
```
</head>

<body>
<form name="form1" method="post" action="">
<input name="Button" type="submit" id="Button" value="Bottez-moi !"
       onClick="alert('Aiiiiiiiie !')">
</form>
</body>
</html>
```

provoquera l'apparition d'une boîte de dialogue lorsque l'on appuiera sur la bouton :



A remarquer cependant que les navigateurs peuvent bloquer les éléments actifs, comme par exemple l'affichage d'une boîte de dialogue ou d'un message dans la barre d'état. En voici un exemple avec Internet Explorer :





9. Un exemple de validation de formulaire

Envisageons un second exemple plus intéressant : dans le contexte d'un formulaire permettant un login assez classique (nom, prénom, password-identifiant choisi, indication du sexe), nous souhaitons éviter d'envoyer au serveur un champ nom et/ou un champ password-identifiant vide. Une fonction JavaScript valideFormulaire() va réaliser ce contrôle :

- ◆ elle vérifie que le 1er champ (donc, le nom) du formulaire passé en paramètre est non vide; si il l'est, un booléen ok est mis à false;
- ◆ idem pour le 3^{ème} champ (le password-id);
- ◆ si le booléen est à true, le formulaire est envoyé au serveur au moyen de la méthode submit() de form (le serveur va appeler une servlet Java nommée TraiteLogin); sinon, un message prévient que le formulaire n'a pas été envoyé.

Le bouton d'envoi du formulaire n'est plus déclaré comme étant de type submit, mais de type button seulement; mais il est muni d'un attribut onClick qui appelle la fonction valideFormulaire() :

login-js1.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Nature++ : Accueil</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

```

<script language="JavaScript" >
<!--
function valideFormulaire(formul)
{
    ok = true;
    if (formul[0].value == "")
    {
        alert("Veuillez remplir le champ nom du formulaire"); ok = false;
    }
    if (formul[2].value == "")
    {
        alert("Veuillez remplir le champ password du formulaire"); ok = false;
    }
    if (ok) formul.submit();
    else alert("Le formulaire n'a pas été envoyé");
}
//-->
</script>
</head>

<body>
<p><strong><font color="#3300FF" size="+2">Ligue des Joyeux Protecteurs de la
Nature et du Naturisme</font></strong></p><hr>

<p><strong><em><font color="#CC0000" size="+1">Bonjour toi !</font></em>
</strong></p>

<p>Qui es-tu ?</p>

<form action="TraiteLogin" method="post" name="formulaireLogin">
<label>Ton nom* :</label>
<input type="text" name="nom">
<p> <label>Ton prénom :</label> <input type="text" name="prenom"> </p>
<p>L'identifiant (password-id)* que tu utiliseras sur ce site :
<input type="password" name="password" > </p>
<p> <label>Homme ou femme ?</label> </p>
<table width="200">
<tr>
<td><label>
<input type="radio" name="GroupeSexe" value="1">
Homme</label></td>
</tr>
<tr>
<td><label>
<input type="radio" name="GroupeSexe" value="2">
Femme</label></td>
</tr>

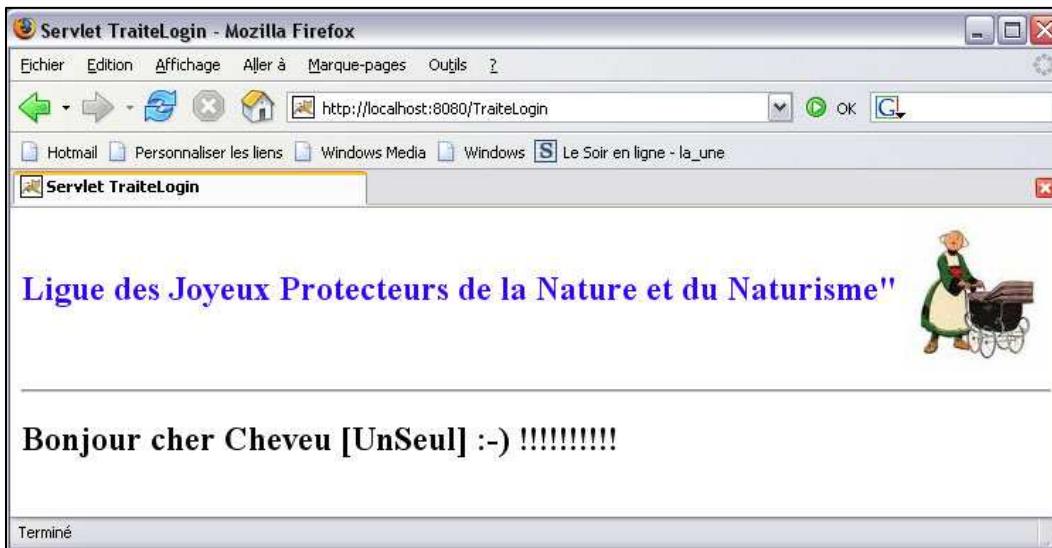
```

```
</table>
<p>
<input type="hidden" name="action" >
&nbsp;
<input type="button" name="Submit" value="Envoyer"
onClick="valideFormulaire(this.form)" >
</p>
</form>

<hr>
<p><em>Derni&egrave;re mise &agrave; jour</em> : samedi, 30/06/07</p>
<p>&nbsp;</p>
</body>
</html>
```

Dans le cas d'un formulaire correctement rempli, on obtient :

The screenshot shows a Mozilla Firefox window displaying a web page titled "Nature++ : Accueil". The URL in the address bar is "http://localhost:8080/login-js1.htm". The page content includes a logo of a woman pushing a stroller, a title "Ligue des Joyeux Protecteurs de la Nature et du Naturisme", and a "Bonjour toi !" greeting. Below this, there are fields for "Ton nom*" (Cheveu), "Ton prénom" (Mathieu), and "L'identifiant (password-id)* que tu utiliseras sur ce site" (password). There are gender selection buttons ("Homme" or "Femme") and an "Envoyer" button. At the bottom, it says "Dernière mise à jour : samedi, 30/06/07" and "Terminé".



avec comme servlet côté serveur :

TraiteLogin.java

```
/*
 * TraiteLogin.java
 */

package servletsjs;

import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;

/**
 * @author Vilvens
 */

public class TraiteLogin extends HttpServlet
{
    protected void processRequest (HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException
    {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        String nomRecu = request.getParameter("nom");
        String pwdRecu = request.getParameter("password");
        String sexeRecu = request.getParameter("GroupeSexe");

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet TraiteLogin</title>");
```

```
out.println("</head>");
out.println("<body>");
String titre = "<p><strong><font color=\"#3300FF\" size=\"+2\">Ligue des Joyeux
Protecteurs de la Nature et du Naturisme</font> <img src=\"becassine.jpg\" width=\"101\"
height=\"104\" align=\"absmiddle\"></strong></p><hr>";
out.println(titre);
out.println("<p><H2>Bonjour </H2><p>");

if (sexeRecu.equals("1")) out.println(" cher ");
else out.println(" chère ");
out.println(nomRecu + " [" + pwdRecu +"] :-) !!!!!!!</H2><p>");

out.println("</body>");
out.println("</html>");

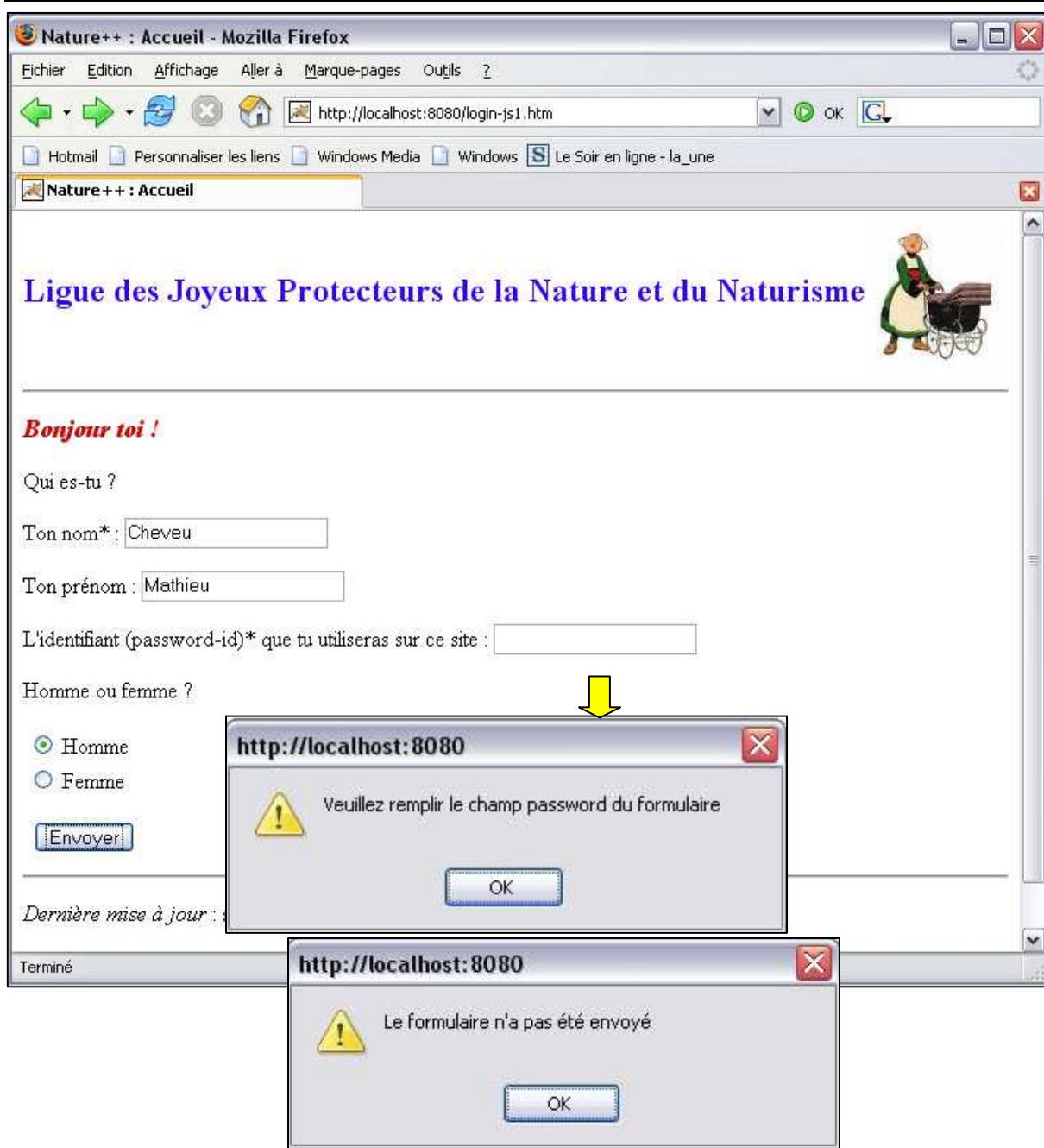
out.close();
}

protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{ processRequest(request, response); }

protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{ processRequest(request, response); }

public String getServletInfo() { return "Servlet pour client JS"; }
```

Mais si l'un des champs exigés est absent :

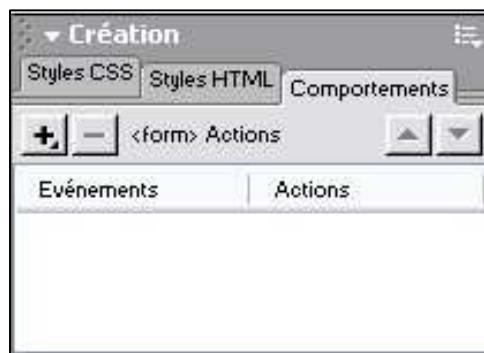


Remarques

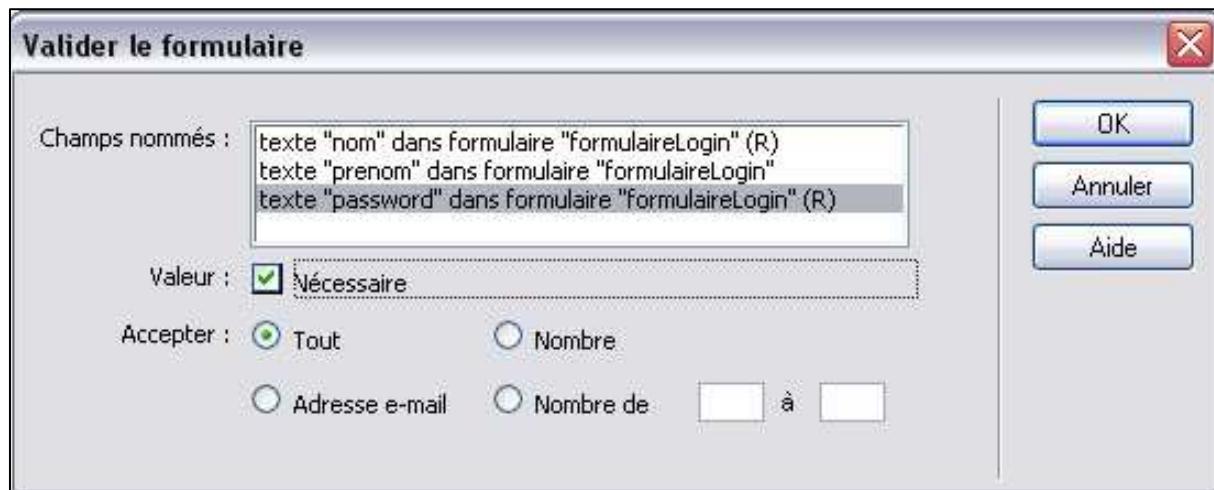
1) Dans le contexte de la validation, il peut être utile de connaître l'existence de fonctions globales (donc non incluses dans une classe ou un objet) :

fonction globale	fonctionnalité
boolean isNaN (String chaîne)	vérifie si une chaîne est un nombre ou pas
int parseInt (String chaîne, [Integer base])	convertit une chaîne en entier – renvoie NaN si la conversion a échoué
float parseFloat (String chaîne)	convertit une chaîne en réel flottant – renvoie NaN si la conversion a échoué
String escape (String chaîne)/ String unescape (String chaîne)	conversion des caractères non ASCII 7bits en leur forme hexadécimale (%XX) ou l'inverse

2) Un logiciel de conception Web comme Dreamweaver permet de générer de telles fonctions de manière simple. En effet, il suffit de sélectionner un élément (ici, le formulaire) et de cliquer le bouton "+" dans l'onglet "Comportements" de la fenêtre "Création" :



On peut alors choisir dans le menu contextuel "Valider le formulaire" pour obtenir :



Mais le résultat généré n'est pas vraiment si simple, par exemple :

```
...
function MM_findObj(n, d) { //v4.01
  var p,i,x; if(!d) d=document; if((p=n.indexOf("?"))>0&&parent.frames.length) {
    d=parent.frames[n.substring(p+1)].document; n=n.substring(0,p);}
  if(!(x=d[n])&&d.all) x=d.all[n]; for (i=0;!x&&i<d.forms.length;i++) x=d.forms[i][n];
  for(i=0;!x&&d.layers&&i<d.layers.length;i++) x=MM_findObj(n,d.layers[i].document);
  if(!x && d.getElementById) x=d.getElementById(n); return x;
}

function MM_validateForm() { //v4.0
  var i,p,q,nm,test,num,min,max,errors='',args=MM_validateForm.arguments;
  for (i=0; i<(args.length-2); i+=3) { test=args[i+2]; val=MM_findObj(args[i]);
  if (val) { nm=val.name; if ((val=val.value)=='') {
    if (test.indexOf('isEmail')!=-1) { p=val.indexOf('@');
      if (p<1 || p==(val.length-1)) errors+='- '+nm+' must contain an e-mail address.\n';
    } else if (test!='R') { num = parseFloat(val);
      if (isNaN(val)) errors+='- '+nm+' must contain a number.\n';
    }
    if (test.indexOf('inRange') != -1) { p=test.indexOf(':');
      min=test.substring(8,p); max=test.substring(p+1);
    }
  }
}
}
}
```

```

if (num<min || max<num) errors+= '- '+nm+' must contain a number between '+min+' and '+max+'.\n';
} } } else if (test.charAt(0) == 'R') errors += '- '+nm+' is required.\n'; }
} if (errors) alert('The following error(s) occurred:\n'+errors);
document.MM_returnValue = (errors == "");
}

<form action="" method="post" name="formulaireLogin" onSubmit="MM_validateForm(  

'nom','R','password','R'); return document.MM_returnValue">  

...

```

Autrement dit, on y use et abuse des méthodes de la classe **String** ...

3) Toujours dans le contexte des manipulations logiques élémentaires, il faut connaître l'existence de quelques classes utiles : outre String, citons **Math**, **Date** et **Array**.

4) Il existe en JavaScript un objet **navigator** dont les propriétés peuvent être précieuses pour connaître la langue de l'utilisateur ou encore savoir les cookies sont acceptés :

navigator	
<input checked="" type="checkbox"/> platform	nom du système d'exploitation
<input checked="" type="checkbox"/> appName	nom du navigateur
<input checked="" type="checkbox"/> userAgent	à votre avis ?
<input checked="" type="checkbox"/> userLanguage	langue de l'utilisateur (sous forme "fr", "en", etc)
<input checked="" type="checkbox"/> cookieEnabled	true ou false selon que le navigateur autorise les cookies ou pas
<input checked="" type="checkbox"/> plugins	pour obtenir la liste des plugins installés

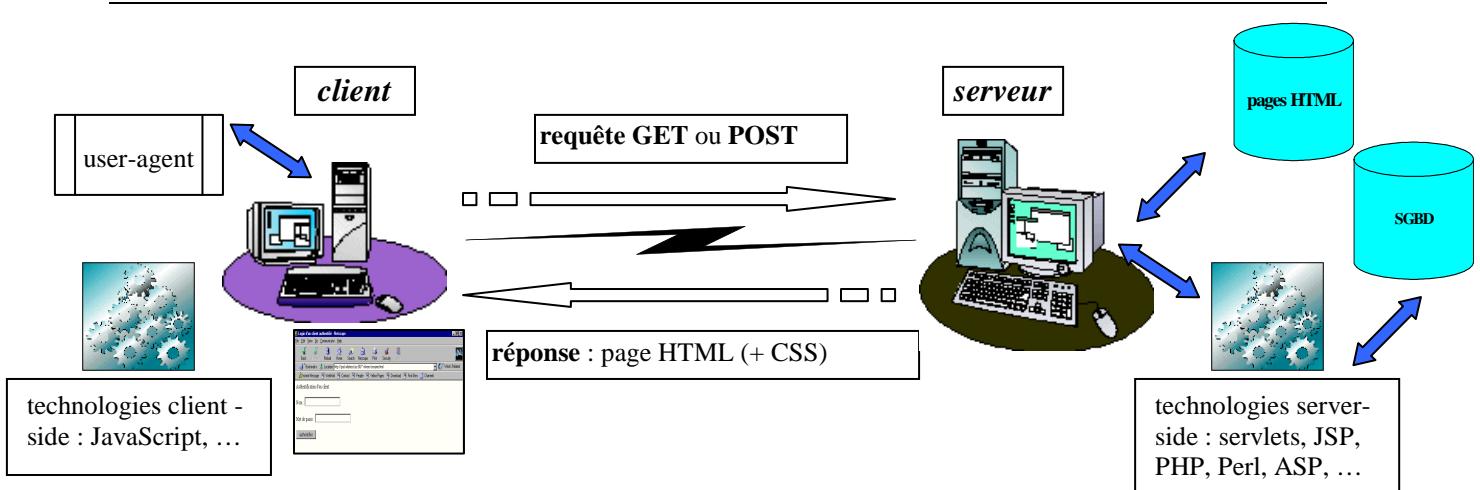
10. Une introduction à AJAX

10.1 Présentation des concepts Web 2.0

Dans le contexte de l'interaction client-serveur Web, le concept **AJAX** (Asynchronous JavaScript Technology and XML) s'est rapidement développé ces derniers mois, notamment sous l'impulsion de Google. En fait, il ne s'agit pas d'une technologie nouvelle, mais plutôt d'un rassemblement de diverses technologies connues qui ont été réunies afin d'accroître le potentiel global :

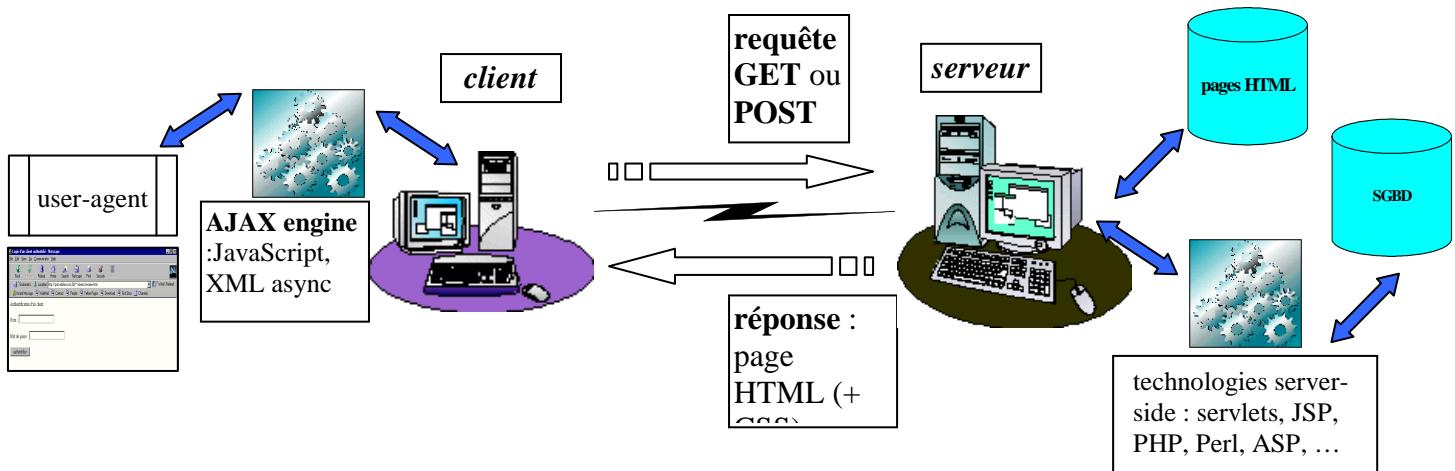
- ◆ le langage du Web XHTML et les styles CSS;
- ◆ XML et les transformations XSLT;
- ◆ la technologie DOM pour XML (DocumentObject Model);
- ◆ JavaScript, en tant que langage de script client;
- ◆ une technique d'asynchronisme au sein de HTTP matérialisé par XMLHttpRequest.

L'idée de base est la suivante. Dans une interaction web-HTTP classique, le client, par l'intermédiaire de son user-agent (c'est-à-dire son browser), envoie une requête au serveur et attend la réponse du serveur. Ce dialogue se répète indéfiniment, avec des temps d'attente pour le client pendant que le serveur effectue les actions appropriées à la requête. On parle encore de "dialogue synchrone".



L'idée d'AJAX est d'introduire une acteur supplémentaire, le "*Ajax Engine*" ("moteur Ajax"), qui va servir d'intermédiaire entre le client et le serveur. Plus précisément, au début d'une session, le user-agent charge cet Ajax Engine (écrit en JavaScript) :

le rôle de celui-ci est de converser avec le serveur et de construire de manière asynchrone l'interface que le browser du client affichera, celui-ci étant ainsi libre de réaliser d'autres tâches sans attendre les réponses du serveur de manière bloquante.



Que gagne-t-on avec une telle architecture ? Que le client peut interagir avec son application client avec un certaine indépendance des communications réseaux avec le serveur, donc sans délai d'attente prohibitif, puisque celles-ci ne sont plus gérées par lui mais par le moteur Ajax. Clairement, l'interface client peut avoir un comportement qui rappelle celui d'une application classique (donc, non Web) : on parle encore de "**rich client**". Donc, l'"Ajax engine"

- ◆ gère l'interactivité avec le client (comme par exemple valider une date ou effectuer un calcul ou une modification de rendu);
 - ◆ réalise les requêtes vers le serveur et reçoit les réponses (ceci au moyen de son composant **XMLHttpRequest**, qui est concrètement un objet JavaScript), ces requêtes réclamant de nouvelles données, demandant l'exécution de processus serveurs, etc.
- On parle cette fois de "dialogue asynchrone".

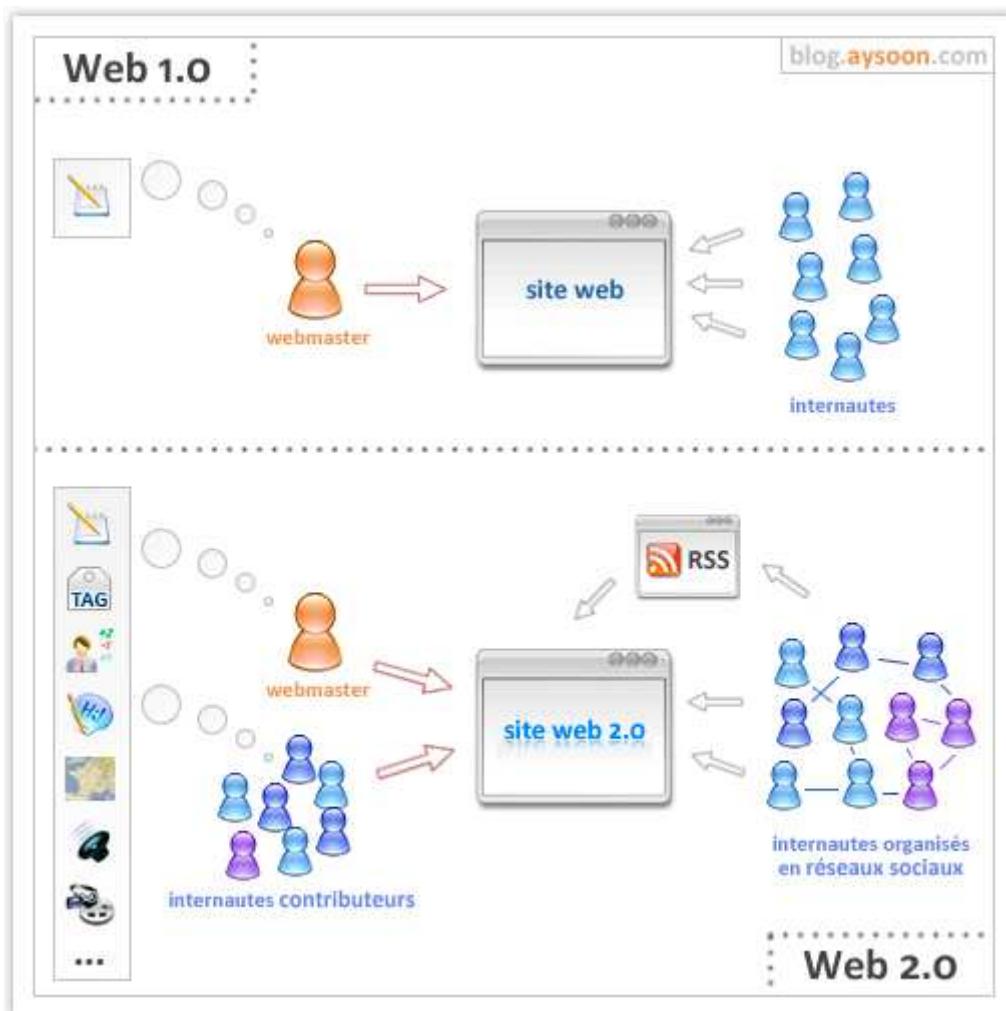
On peut donc constater qu'il n'y pas vraiment d'utilisation de technologie nouvelle : l'évolution se situe plutôt dans la manière de penser les "applications Web", en tenant compte

de cette nouvelle interactivité avec le client. En effet, on peut concevoir que le client dispose au départ d'un container Web de type HTML dans lequel vont s'exprimer un certains nombre de composants (souvent de type XML) : le rôle du serveur ne sera donc plus de fournir systématiquement une page HTML complète, mais plutôt tel ou tel composant demandé par le moteur Ajax – dans ces conditions, le rafraîchissement d'une page dans le browser n'implique plus le chargement de toute la page, mais seulement des composantes modifiées.

AJAX est l'une des pierres d'angle de ce qu'il est convenu d'appeler le Web 2.0, concept bien flou mais qui veut marquer un changement radical dans les applications Web en s'appuyant sur

- ◆ des technologies bien connues mais, jusqu'à ces derniers temps, pas vraiment utilisées conjointement : HTTP et (X)HTML, URIs, CSS, JavaScript, XML (avec DOM), RSS, services Web, ...
- ◆ des méthodes d'utilisation nouvelles basées sur un caractère interactif et collaboratif : blogs, wikis, systèmes de "tagging" pour catégoriser les ressources Web, réseaux sociaux, ...

Le blog <http://blog.aysoon.com/Le-Web20-illustre-en-une-seule-image> résume l'opposition entre le Web 1.0 (le "Web zapette") le Web 2.0 de cette manière :



tandis que cette autre vision des choses est plus "utilisatrice" :



10.2 Un modeste exemple Ajax synchrone

La puissance d'Ajax se révèle pleinement en conjonction avec des technologies serveur que nous verrons ultérieurement (CGI, servlets et Java Server Page, etc). On peut cependant appréhender la philosophie générale avec le modeste exemple suivant.

1) Le client démarra avec une page HTML simpliste :

accueil-ajax.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
    <title>Site Web 2.0 : Accueil</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>

<script type="text/javascript" src="accueil.js"></script>

Accueil
<p><a href="javascript:ajax();">Petit message :-)</a></p>

</body>
</html>
```



L'"événement client" est donc le clic sur le lien (ce pourrait être autre chose, comme une réponse clavier ou souris sur un interface).

2) Un objet XMLHttpRequest est créé et configuré :

accueil.js (version synchrone)

```
// JavaScript Document
function ajax()
{
    var req=null;

    if (window.XMLHttpRequest)
    {
        req = new XMLHttpRequest();
    }
    else if (window.ActiveXObject)
    {
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
    //on demande le fichier reponse.txt
    req.open("GET", "http://localhost:8080/reponse.txt", false);
    req.send(null);

    window.alert(req.responseText);
}
```

La requête est envoyée au serveur au moyen de l'API :

```
open (String method, String URL, boolean asynchronous)
```

dont la signification des paramètres est bien claire. Ici, nous ne demandons pas le mode asynchrone, mais nous y reviendrons.

3) Lorsque l'instruction

```
req.send(null);
```

est atteinte, la requête est envoyée au serveur – ici, un simple accès à un fichier txt, mais il pourrait s'agir d'un appel à une servlet ou un JSP (par exemple ;-)).

4) Le serveur traite la demande – dans notre cas, cherche la page et l'envoie (avec une servlet, c'est celle-ci qui fabriquera la réponse).

5) L'objet XMLHttpRequest attend la réponse et affiche finalement une MessageBox :



Evidemment, jusque là, on ne gagne pas grand' chose

10.3 Un modeste exemple Ajax asynchrone texte

Nous allons reprendre le dialogue précédent, mais en **mode asynchrone** :

```
req.open("GET", "http://localhost:8080/reponse.txt", true);
```

Il nous faut alors mettre en place une fonction callback qui sera appelée lorsque la réponse du serveur arrivera. On peut connaître *l'état d'avancement du traitement de la réponse du serveur* au moyen d'une variable readyState qui peut prendre les valeurs suivantes :

valeur de readyState	traitement de la réponse
0	non initialisé
1	réponse en chargement
2	réponse chargée
3	réponse en cours de traitement
4	traitement terminé

On peut alors gérer l'événement de changement de la valeur de cette variable d'état selon :

```
<requête>.onreadystatechange = <function callback>
```

Notre fichier JavaScript ressemble donc à présent à ceci :

accueil.js (version asynchrone avec réponse texte)
<pre>// JavaScript Document function ajax() { var req=null; if (window.XMLHttpRequest) { req = new XMLHttpRequest(); } }</pre>

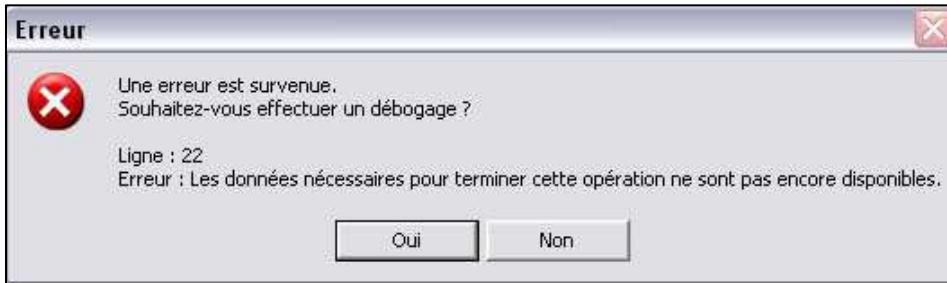
```
else if (window.ActiveXObject)
{
    req = new ActiveXObject("Microsoft.XMLHTTP");
}
//on appelle le fichier reponse.txt
req.open("GET", "http://localhost:8080/reponse.txt", true);
req.onreadystatechange = function() { callback(req); }
req.send(null);
}

function callback(req)
{
    if (req.readyState == 4)
    {
        if (req.status == 200)
        {
            window.alert(req.responseText);
        }
    }
}
```

Avec une page HTML et un fichier texte de réponse dont on a changé le message, cela donne :



Si nous n'avions pas testé la valeur de la variable membre readyState de la requête, nous n'aurions obtenu la boîte de dialogue ci-dessus qu'après des apparitions successives de :



Logique : à chaque changement de valeur de 0 à 3, la réponse n'est pas encore complètement traitée ...

10.4 Un modeste exemple Ajax asynchrone xml

Nous en arrivons enfin réellement à l'ensemble des principes Ajax. En fait, les browsers gèrent une représentation XML de ce qu'ils proposent au client. Ce document XML est géré selon DOM : une modification quelconque est donc toujours possible.

Notre réponse va à présent prendre la forme d'un (très modeste) fichier xml :

reponse.xml

```
<?xml version="1.0"?>
<message>Bonjour</message>
```

La fonction callback reçoit donc à présent un document xml (au sens DOM) : elle y rechercher les informations qu'elle souhaite – dans notre cas, le contenu du tag message.

accueil.js (version asynchrone avec réponse xml)

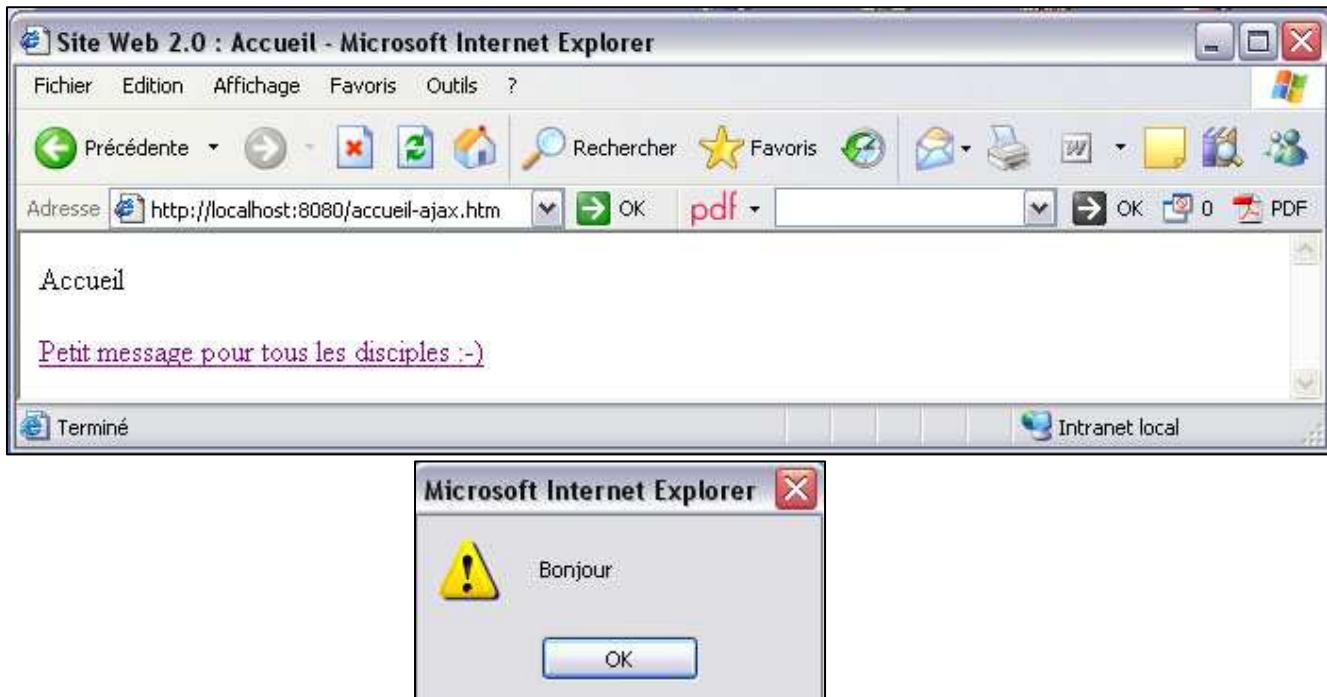
```
// JavaScript Document
function ajax()
{
    var req=null;

    if (window.XMLHttpRequest)
    {
        req = new XMLHttpRequest();
    }
    else if (window.ActiveXObject)
    {
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
    req.open("GET", "http://localhost:8080/reponse.xml", true);
    req.onreadystatechange = function() { callback(req); }
    req.send(null);
}

function callback(req)
{
    if (req.readyState == 4)
    {
        if (req.status == 200)
```

```
{  
    var message = req.responseXML.getElementsByTagName("message")[0];  
    alert(message.childNodes[0].nodeValue);  
}  
}
```

Avec une page HTML dont on a changé le message, cela donne :



Bien sûr, notre exemple est ici ultra-élémentaire. La démarche complète est plutôt que :

- ◆ le fichier XML est confectionné côté serveur par une **servlet Java** (par exemple – bref par un technologie server side), en fonction de la nature de la requête;
- ◆ la page HTML du client est mise à jour par la fonction callback au moyen de la propriété **innerHTML**, qui comporte l'arbre DOM reflétant la page HTML du client : il est donc parfaitement possible de modifier celui-ci, y compris en ajoutant de nouveaux éléments.



JavaScript joue donc un rôle fort intéressant côté client, notamment pour les contrôles. Et pourtant, HTML pourra bientôt se passer de JavaScript pour ce genre de choses – et pour bien d'autres ...

X. Quelques nouveautés de HTML 5



L'intelligence sans humour est difficilement de la vraie intelligence.

(E. Chatiliez, Ciné Live – 12/ 2001)

1. Encore une nouvelle norme ?

C'est en 1998 que le W3C décida de ne plus faire évoluer le HTML dans sa définition native, avec toute la permissivité que cela impliquait comme par exemple le fait qu'un tag n'était pas forcément suivi par son anti-tag. HTML fut figé en sa version 4.01 et le W3C décida de poursuivre l'évolution avec la spécification **XHTML**. Comme expliqué précédemment, XHTML décrit des pages dans un HTML défini comme une application de XML, satisfaisant à une DTD laquelle garantit une syntaxe plus stricte et plus clairement définie. Ainsi, par exemple, tous les tags et tous les attributs doivent être écrits en caractères minuscules, les valeurs des attributs doivent toujours être placées entre guillemets, les tags doivent être emboîtés correctement, tout tag non vide doit être refermé, etc.

Mais un certain nombre de développeurs Web n'étaient pas convaincus que la voie spartiate de XML était la bonne : après tout, c'était précisément parce que les browsers négligeaient tout simplement des séquences non comprises ou incohérentes sans message d'erreur que HTML avait conquis tant de développeurs de sites Web. Rapidement se constitua un groupe **WHATWG** (**Web Hypertext Application Technology Working Group**), formé de développeurs d'Opera, de Mozilla et d'Apple, dont le but était de développer une autre spécification de HTML dans un sens plus conciliant pour le travail de développement quotidien : par exemple, utiliser ou ne pas utiliser de guillemets autour des valeurs des attributs serait laissé à l'appréciation du développeur Web (et il pourrait changer d'avis en cours de route !). Autrement dit : plus de pragmatisme pour "ne pas casser le Web".

C'est cette spécification qui allait devenir **HTML 5** : en 2009, le W3C cessa de travailler sur XHTML 2.0 pour rejoindre le WHATWG. HTML 5 est depuis lors en cours de finalisation (on estime que la version finale apparaîtra en 2014), avec des modifications fréquentes (en maintenant la compatibilité avec l'existant) et des navigateurs qui sont, en 2011, encore loin d'implémenter le support de toutes les possibilités de HTML 5 (et plus souvent qu'à tout : Internet Explorer ;-)) : patience donc !



One Web **W3C** for All

2. Principales nouveautés apportées par HTML 5

En synthèse, on peut épingle les caractéristiques suivantes :

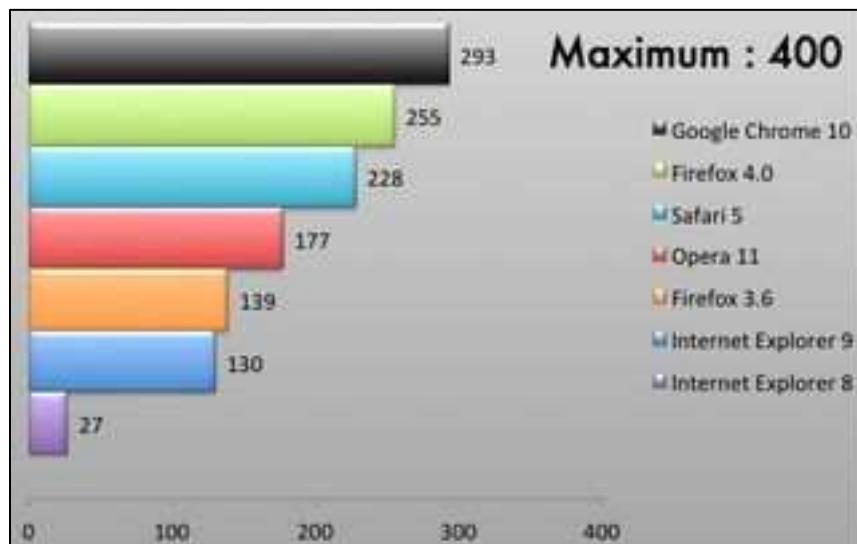
- ◆ capacité à s'adapter automatiquement aux caractéristiques techniques du poste client (PC, Mac, smartphone, tablette, ...) et de déléguer une partie du travail aux processeurs disponibles;
- ◆ simplification de tags comme `<doctype>`, `<meta>`, `<head>`, etc;
- ◆ apport de nouveaux tags "sémantiques" comme `<article>`, `<header>`, `<footer>`, `<nav>`, etc ... qui se présentent comme une spécialisation du tag `<div>`;

- ◆ faculté pour les formulaires de contrôler les entrées (donc sans le concours de fonctions JavaScript) grâce à de nouveaux attributs de la balise `<input>` : date, week, email, url, range, etc;
- ◆ ajout de nombreux nouveaux attributs pour de multiples balises et extension d'attributs existants à des balises qui ne les admettaient pas auparavant;
- ◆ abandon des frames, donc de tous leurs tags et attributs;
- ◆ balises `<video>` et `<audio>` dédiées à la diffusion multimédia;
- ◆ élément `<canvas>` qui fournit une énorme API (JavaScript) dédiée aux dessins à 2 dimensions;
- ◆ gestion de la consultation en mode déconnecté au moyen d'un cache local;
- ◆ remplacement à terme des cookies par l'API (JavaScript) des éléments `sessionStorage` et `localStorage`;
- ◆ gestion du couper-coller;
- ◆ API de messagerie électronique étendue;
- ◆ ouverture à la spécification séparée des "Web workers" qui sont en fait des threads pour Java script.

Répétons-le, les navigateurs de 2011 sont loin de supporter tous les features de HTML :



mais l'évolution est encourageante comme le prouve ces résultats au test officiel de conformité à HTML5 (400 points en cas de conformité totale) :



(source : <http://www.zdnet.fr/blogs/entreprise-2-0/html5-clef-de-la-reussite-du-cloud-computing-39760020.htm>)

Inutile dire aussi qu'avec de telles possibilités, HTML 5 hisse les applications Web à un niveau compatible avec les exigences et les potentialités du Cloud computing.

Dans cette introduction, nous allons illustrer deux fonctionnalités simples de HTML 5 : les balises "sémantiques" de navigation au sein d'une page et la validation des champs de formulaires.

3. Le remplacement des frames : les balises sémantiques de navigation

Nous avons déjà vu que la balise `<div>` permettait d'obtenir l'équivalent d'une présentation en frames, mais sur un seul document et donc finalement de manière plus simple. Prenons l'exemple suivant :

blog01.htm

```
<!DOCTYPE HTML>
<meta charset=utf-8; charset=iso-8859-1 >

<html>

<head>
<title>Blog d'un génie</title>
<link href="blog.css" type=text/css rel="stylesheet" media="screen">
</head>

<body>
<p>Tout petit, je rêvais d'être moi - et mon rêve d'enfant s'est réalisé ;-)</p>
<div id=entete class=titre>
<h2>Hellooooooooooooo ... </h2>
</div>
<div id=menu>
Consulter :
<ul>
```

```
<li><a href=semaine-derniere.html>Semaine dernière</a></li>
<li><a href=liste-messages>Liste complète</a></li>
</ul>
</div>
<div class=poste>
<h2>Humeur</h2>
<p>Pourquoi le Standard vend-il tous ses bons joueurs ? Pour être sûr de ne jamais être champion ?</p>
</div>
<div class=poste>
<h2>Tristesse</h2>
<p>L'été 2011 ? Une horreur sans soleil !</p>
</div>
<div id=pied>
<p><small>(c) Vilvens Claude</small></p>
</div>
</body>
</html>
```

On remarquera d'emblée, et principalement :

- ◆ le DOCTYPE simplifié :-)
- ◆ le tag meta placé hors de la zone head
- ◆ l'absence de guillemets (mais on aurait pu les utiliser).

Le document CSS associé est simplement :

blog.css

```
#menu { float:left; width:20%; border-style:solid; border-width:medium; border-color:lightgreen }
.poste { float:right; width:79%; background-color:lightgreen; }
.titre { background-color:lightblue; border-style:double; }
#pied { clear:both; border-style:dotted; border-width:thin; border-color:red }
```

et le résultat est bien logiquement :



HTML 5 va apporter les balises de navigation `<nav>`, `<header>`, etc à la signification bien claire et qui vont remplacer l'utilisation générique de `<div>`. On peut ainsi imaginer :

blog02.htm

```
<!DOCTYPE HTML>
<meta charset=utf-8><!-- charset=iso-8859-1-->
<html>

<head>
<title>Blog d'un génie</title>
<link href=blog02.css type=text/css rel="stylesheet" media="screen">
</head>

<body>
<p>Tout petit, je rêvais d'être moi - et mon rêve d'enfant s'est réalisé ;-)</p>
<header>
<!-- class="titre"-->
<h2>Helloooooooooooooo ... </h2>
</header>
<nav>
Consultez :
<ul>
<li><a href=semaine-derniere.html>Semaine dernière</a></li>
<li><a href=liste-messages>Liste complète</a></li>
</ul>
</nav>
<article>
```

```
<h2>Humeur</h2>
<p>Pourquoi le Standard vend-il tous ses bons joueurs ? Pour être sûr de ne jamais être champion ?</p>
</article>
<article>
<h2>Tristesse</h2>
<p>L'été 2011 ? Une horreur sans soleil !</p>
</article>
<footer>
<p><small>(c) Vilvens Claude</small></p>
</footer>
</body>
</html>
```

Il se peut que la version du navigateur utilisée ne connaisse pas encore les balises en question. Mais, même dans ce cas, il suffit d'ajouter la feuille de style qui indique que toutes ces balises sont des blocs et comment elles se placent :

blog02.css

```
header, nav, footer, article {display:block;}
nav {float:left; width:20%;}
article {float:right; width:79%; }
footer {clear:both; }
```

On obtient ainsi :



Donc, pas de problème avec Firefox – avec Google Chrome non plus. Par contre, cela ne fonctionne pas avec Internet Explorer ☺ ... Pourquoi ? Personne ne sait vraiment, mais la

solution est bien connue et traîne sur Internet : il faut définir les éléments par JavaScript, donc :

blog02.htm

```
<!DOCTYPE HTML>
<meta charset=utf-8><!-- charset=iso-8859-1-->
<html>

<head>
<title>Blog d'un génie</title>
<link href=blog02.css type=text/css rel="stylesheet" media="screen">
</head>

<script>
document.createElement('header');
document.createElement('nav');
document.createElement('article');
document.createElement('footer');
</script>

<body>
<p>Tout petit, je rêvais d'être moi - et mon rêve d'enfant s'est réalisé ;-)</p>
<header><!-- class="titre"--><h2>Hellooooooooooooo ... </h2></header>
<nav> ...
</body>
</html>
```

Et tout fonctionne ! On remarquera en passant que la balise `<script>` sans attribut suffit ...

4. Les formulaires avec validation de champs

Le problème est bien connu : en HTML classique, rien n'est prévu côté client pour contrôler le contenu des champs `<input>` d'un formulaire. On peut évidemment contourner le problème avec des fonctions Javascript ou des applets Java (par exemple), mais HTML 5 offre plus simple : de nouveaux type de champs `<input>` avec contrôle intégré. Ainsi en est-il des types email, url, date, month, week, number, range, color, ... On peut donc imaginer ceci :

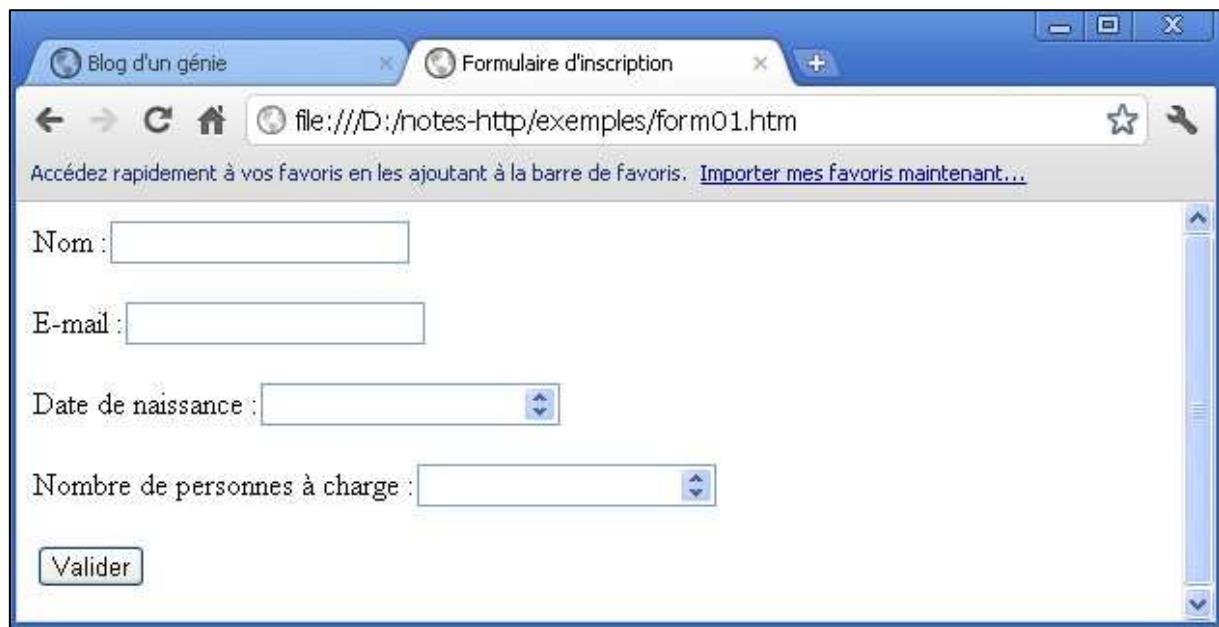
form01.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<meta charset=utf-8; charset=iso-8859-1 >
<html>
<head>
<title>Formulaire d'inscription</title>
</head>

<body>
<form>
<p>Nom :<input name=nom type=text required>
</p>
```

```
<p>E-mail :<input name=email type=email required>
</p>
<p>Date de naissance :<input name=dnais type=date required>
</p>
<p>Nombre de personnes à charge :<input name=npc type=number min=0 max=20 step=1 required>
</p>
<P><input type=submit ...></P>
</form>
</body>
</html>
```

Aucun effet particulier sous Firefox ⓘ - mais avec Google Chrome :



On constate d'emblée la forme des champs de type date et nombre qui, par un mécanisme de combo, proposent les valeurs possibles (sous Opera, la date peut même apparaître sous forme d'un contrôle du genre JXDatePicker de Java (par exemple). Les contrôles sont bien effectifs :

The following table summarizes the data entered in each screenshot:

Screenshot	Nom	E-mail	Date de naissance	Nombre de personnes à charge
1				
2	Vilvens Denys	j'en ai pas		
3	Vilvens Denys	denys.vilvens@guitar.net	j'ai oublié	
4	Vilvens Denys	denys.vilvens@guitar.net	1993-12-19	450
5	Vilvens Denys	denys.vilvens@guitar.net	1993-12-19	2

Allez, on y est arrivé :

The screenshot shows a Windows-style window for a web browser. The title bar says "Blog d'un génie" and "Formulaire d'inscription". The address bar shows "file:///D:/notes-http/exemples/form01.htm?nom=Vilvens+Denys&email=denys.vilvens%40guitar.net". Below the address bar, there's a message: "Accédez rapidement à vos favoris en les ajoutant à la barre de favoris. [Importer mes favoris maintenant...](#)". The main content area contains four input fields: "Nom:" with a text box, "E-mail:" with a text box, "Date de naissance:" with a date picker, and "Nombre de personnes à charge:" with a dropdown menu. At the bottom left is a "Valider" button.



On l'a compris, nous pourrions poursuivre. De plus, on sait que les pages WEB peuvent être également améliorées en utilisant des **applets** Java – ceci fait l'objet d'un chapitre d'un cours de Java. Cependant, on ne peut ignorer Java lorsqu'il s'agit de remplacer les CGIs par des **servlets** ou des **Java Server Pages** – ce qui fait l'objet d'autres chapitres d'un cours de Java ... Sans parler de **PHP** et des **ASP.NET** ;-) ! C'est sûr : HTTP will return ;-)

Ouvrages consultés



Ouvrages imprimés

Avedal, K, et al. Professional JSP. Birmingham, United Kingdom. Wrox Press Ltd. 2000.

Campione, M., Walrath, K., Huml, A & the tutorial team. The Java Tutorial Continued / The Java Series. Reading, Massachusetts, U.S.A. Addison-Wesley Publishing Company. 1998.

Cheng, H. H. CGI Programming in C. In : *C/C++ Users Journal*. Vol. 15, N° 1, janvier 1997.

Dufour, A. Internet. Paris, France. Presses Universitaires de France, collection "Que sais-je ?". 2005(1995).

Garance, D. & Houste, F. Macromedia Dreamweaver MX. Campus Press, Paris. 2004.

Grenson, M. Analyse et développement d'architectures WEB et d'un module Java de statistiques (intégrés à un outil de gestion de contenu XML). Seraing, Belgique. TFE In.Pr.E.S. 2004.

Lawson, B & Sharp, R. Introduction à HTML 5. Pearson Education France, Paris. 2011.

Lemay, L. Créer un serveur WEB en HTML. Paris, Ed. Simon & Schuster Macmillan, 1996.

Loshin, P. Essential Email Standards. NewYork, U.S.A. John Wiley & Son, Inc. 2000.

Netscape communications server. Programmer's guide. Netscape Communications Corporation. Mountain View (U.S.A.), 1996.

Vanden Berghen, C. Tout connaître sans oser le demander. In: *Athena*, N° 239-240-241, mars-mai 2008.

Sites Internet

(juillet 2011)

- ◆ <http://www.cs.tut.fi/~jkorpela/http.html>
Un guide de référence sur les headers de http.

- ◆ <http://xmlfr.org/actualites/decid/051201-0001>
Synthèse sur la définition du Web 2.0.
- ◆ <http://www.toutjavascript.com/referenc/>
Une référence du langage JavaScript.
- ◆ <http://pbnaigeon.developpez.com/tutoriel/CSS-HTML/mise-en-page-CSS/>
Pour une explication plus détaillée des CSS avec les div.
- ◆ <http://www.adaptivepath.com/publications/essays/archives/000385.php>
J.J. Garrett. Ajax: A New Approach to Web Applications. Pour une présentation directe et simple d'AJAX.
- ◆ <http://java.sun.com/developer/technicalArticles/J2EE/AJAX/>
G. Murray. Asynchronous JavaScript Technology and XML (AJAX) With the Java Platform. Pour une intégration de la technologie des servlets avec AJAX.
- ◆ <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
R. T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. Une réflexion approfondie sur le design des applications réseaux.
- ◆ <http://www.20thingsilearned.com/fr-FR/web-apps/3>
Introduction sympa à l'Internet du 21^{ème} siècle.
- ◆ <http://w3c.brutoweb.net/html5-diff/2008-06-10.html>
Les différences de HTML 5 par rapport à HTML 4 – par le W3C !