Table des matières

Avant-propos

Ce document rassemble des *notes de cours* rédigées pour la première édition de ce cours, en 1997, et mises à jour en fonction de l'évolution du domaine : le langage HTML s'est enrichi de certaines constructions, notamment les styles et les couches, et le langage JavaScript ne cesse d'évoluer.

Ce cours montre l'un des développements les plus spectaculaires de l'Informatique contemporaine. Leur réalisation nécessite cependant l'apprentissage de plusieurs langages, et la compréhension sommaire de plusieurs protocoles. Il est absolument indispensable de consulter régulièrement les manuels de référence de ces langages, dont les versions en ligne se trouvent évidemment sur le Web aux adresses

- `http://cermics.enpc.fr/doc/html/ref`
- `http://cermics.enpc.fr/doc/javascript/ref`

René Lalement
Champs-sur-Marne, 5 janvier 1999

Chapitre 1

Le Web et l'Internet

Le contexte

- intégration des télécommunications et de l'informatique au moyen des technologies numériques qui gagnent tous les domaines de l'information (téléphone, son, image), désormais multimédia
- changements d'échelle de l'informatique : architecture (réseau mondial), taille des données traitées, nombre d'utilisateurs
- nouveaux impératifs : sécurité et temps réel pour tout le monde

1.1 Le World Wide Web

Le *World Wide Web* [1], conçu au CERN, est le dernier en date des services offerts par l'Internet, et celui qui a la plus forte croissance. C'est un *système d'information hypermédia distribué multiprotocole.*

Un système d'information *multimédia* est composé de plusieurs formats de données : texte, son, image fixes ou animées, bases de données. Un système d'information *hypermédia* est un ensemble de documents multimédias organisé pour pouvoir *naviguer* d'un document à un autre par un simple pointage sur une partie d'un document (en général avec une souris).

Le caractère *distribué* signifie qu'un ensemble de documents peut être réparti physiquement sur plusieurs machines, dites *serveuses.*

Le Web est un système *multiprotocole* dans la mesure où il intègre le protocole HTTP et d'autres protocoles d'application (FTP, Gopher, Telnet, Wais, etc).

1. aussi appelé le Web, WWW, W^3, ou la *Toile*

Un document WWW est composé de *pages HTML* (*Hyper Text Marking Language*) et d'autres documents non-textuels. Un ensemble de documents WWW ayant un même point d'entrée est appelé un *site WWW*.

La visualisation d'un document Web, et la navigation d'un document à un autre se fait à l'aide d'un programme appelé *navigateur* (ou *browser*, *fureteur*, *fouineur*,...) comme Netscape, MS Explorer ou Lynx (dont l'interface est alphanumérique). Le navigateur communique avec d'autres programmes, dits *serveurs*, qui résident sur les machines serveuses.

L'idée du WWW est due à Tim Berners-lee dans un rapport du CERN de mars 1989. La première implémentation date de novembre 1990. Les premiers navigateurs et serveurs sont distribués courant 1991. NCSA distribue son serveur HTTP fin 1992 et le navigateur Mosaic en février 1993. Marc Andreessen et ses collègues quittent NCSA en mars 1994 pour fonder "Mosaic Communications Corp" qui deviendra Netscape en novembre. Le W^3C (World Wide Web Consortium), organisme international industriel, a été fondé en 1994 par le MIT, auquel se sont associés l'INRIA et l'Université Keio, pour développer des protocoles et des codes de référence assurant l'évolution du World Wide Web ; il regroupe plus de cent membres, qui sont des organismes de recherche ou des compagnies. Son site `<http://www.w3.org/>` est la meilleure source d'information sur le Web.

1.2 Désignation des ressources : les URI

Un *Uniform Resource Identifier* (URI), permet de désigner une ressource accessible sur le WWW (un document ou un service).

Un URI est composé de plusieurs parties qui définissent

- le schéma d'accès : les plus utiles sont `http`, `ftp`, `file`, `telnet`, `mailto`, `javascript`
- le chemin d'accès à cette ressource

Dans le cas où cette ressource est un document, l'URI s'appelle *Uniform Resource Locator* (URL) ; le chemin d'accès au document est alors spécifié par le chemin d'accès à la machine serveuse, éventuellement le port, et le chemin d'accès au document sur cette serveuse.

Par exemple `http://cermics.enpc.fr/equipes/proba.html` désigne un document accessible par le schéma `http` sur la machine `cermics.enpc.fr`, dans le fichier de chemin `/equipes/proba.html` ; ce chemin est relatif à un répertoire particulier, appelé *racine WWW du serveur*, où sont regroupés les documents WWW publics sur cette machine. L'emplacement de la racine WWW dépend de l'installation du serveur.

En général, chaque utilisateur peut installer ses propres documents WWW dans un répertoire, appelé *racine WWW de l'utilisateur*, qui se trouve dans son répertoire personnel ; le nom de cette racine dépend également du serveur, mais est le même pour tous les utilisateurs (c'est souvent `www`). Pour ces documents personnels, l'URL indique ~*login* entre la machine et le fichier, par exemple :

`http://cermics.enpc.fr/~dh/fromage/big_cheese.html`

Le chemin `fromage/big_cheese.html` est relatif à la racine WWW de l'utilisateur `dh`.

Un URL sert à désigner un document à partir d'un autre document. Si ces deux documents sont sur le même serveur, on peut utiliser un URL local, où on ne spécifie ni le schéma d'accès ni la machine ; on écrit seulement `/equipes/proba.html` (le `/` initial est indispensable car il indique un chemin relatif à la racine WWW). Il est aussi possible de spécifier un chemin relatif à un autre document, par exemple `../reports/index.html` ; dans ce cas, l'URL ne comporte pas de `/` initial.

1.3 Les services d'application

Un service est mis en œuvre par un ensemble de programmes partageant un même protocole.

Un *protocole* est un ensemble de règles (format de données, de dialogues, etc) permettant de réaliser un service. Un protocole n'est pas un programme, mais il est implémenté par un ensemble de programmes.

Les services d'application fonctionnent généralement sur le mode *client-serveur*. Clients et serveurs sont des processus[2]. Un service d'application se compose ainsi de deux processus, chacun s'exécutant sur une machine et communicant par l'intermédiaire d'un protocole.

Le processus client prend l'initiative d'une communication destinée au processus serveur et lui transmet une requête. Ce processus serveur écoute en permanence les requêtes : il peut leur répondre lui-même, ou créer un autre processus serveur qu'il charge de cette réponse. Les serveurs sont souvent des *démons* : on trouve par exemple les programmes `ftpd`, `telnetd` et `httpd`, réalisant les services FTP, Telnet et HTTP sous Unix. Pour ces mêmes applications, on peut trouver côté client les programmes `ftp`, `telnet`, `netscape`, également sous Unix.

Dans le cas du Web, le protocole est **HTTP**, « HyperText Transfer Protocol » version 1.1 [RFC 2068], les programmes clients sont des *navigateurs*, comme Netscape Navigator, Microsoft Explorer ou Lynx (University of Kansas) ; le programme serveur est généralement `httpd` (sous Unix).

Les premières utilisations de l'Internet reposaient sur les trois applications suivantes :

– connexion à distance (protocole Telnet [RFC 854, 855])

2. un processus est un programme en cours d'exécution ; un même programme peut donner lieu à plusieurs processus coexistants sur la même machine, chacun ayant sa propre zone de mémoire.

– transfert de fichiers (protocole FTP [RFC 959])
– courrier électronique (protocole SMTP [RFC 821, 1869] et format des messages [RFC 822])

Notons que le courrier électronique est à la base des *listes de distribution*. Ce sont des listes d'adresses électroniques dédiées à un sujet particulier ; il en existe des milliers. Chaque liste a une adresse de gestion et une adresse de communication. On devient membre d'une liste en envoyant un message de souscription à l'adresse de gestion de la liste. La liste des membres de la liste n'est en général connue que de celui qui la gère. Tout membre peut faire parvenir un message à tous les autres membres de la liste, en envoyant un message à l'adresse de communication de la liste.

L'idée des *News* (en français *forum électronique*) est voisine des listes de distribution, mais l'organisation est différente. Ce ne sont pas des individus qui s'inscrivent sur une liste, mais des machines qui sont organisées pour transmettre aux autres les articles qu'elles reçoivent. L'ensemble de ces machines, appelées serveuses de News, constitue le réseau Usenet. Les "articles" sont classés par groupe thématique, ou *newsgroup*. Il existe plusieurs centaines de groupes, dont la plupart ont un domaine de distribution mondial ; un site ou un ensemble de sites peuvent aussi créer des groupes dont le distribution est local ou régional. Une machine ne reçoit pas nécessairement tous ces groupes, mais seulement une sélection qui a été demandée par son administrateur. Un serveur de News propage chaque article posté localement aux machines appartenant à son domaine de distribution. Entre deux machines connectées par l'Internet, cette propagation se fait à l'aide du protocole NNTP « Network News Transfer Protocol » [RFC 977].

Les trois services à l'origine de l'Internet ont rapidement été utilisés pour l'accès à l'information :

– connexion à distance : pour accéder à des bases de données (notamment les catalogues des bibliothèques)
– transfert de fichiers : pour télécharger des documents (par exemple les RFC d'Internet) et des logiciels
– courrier électronique : pour demander l'envoi de documents et de logiciels, pour ceux qui ne disposent que de ce moyen

Ils ont été suivis d'autres services, Gopher, WAIS, le World Wide Web, qui ont rompu avec le schéma centralisé de l'information. En effet, l'Internet permet naturellement une organisation *distribuée* de l'information, ce qui signifie qu'un ensemble de documents peut être réparti physiquement sur plusieurs machines, appelées *serveurs*. Un service distribué d'information se compose à la fois d'outils pour transférer l'information (dont le prototype est FTP) et d'outils pour localiser l'information, des *annuaires*. Ces derniers sont essentiels à l'utilisation de l'Internet en tant que "base mondiale d'information".

Le premier système réalisant cette idée fut *Gopher*, dû à l'Université du Minnesota. L'information est structurée sous la forme d'une arborescence, l'accès se faisant à chaque niveau en sélectionnant un item dans un menu, jusqu'au document recherché qui figure comme une feuille de l'arborescence. Le service d'annuaire associé à Gopher est *Veronica*. Les serveurs Gopher existent encore, mais sont rendus obsolètes par le développement du World Wide Web.

Le service d'annuaire associé à FTP a longtemps été *Archie*, qui est un autre service d'application. Un annuaire FTP contient des listes de serveurs FTP anonymes, et pour chacun de ces serveurs, des listes du contenu de leurs répertoires. En interrogeant un tel annuaire, on peut savoir sur quel serveur FTP et dans lequel de ses répertoires se trouve le fichier recherché. L'annuaire couramment utilisé est FTP Search de Lycos.

Il existe aussi des serveurs FTP qui rassemblent une masse considérable de données et de logiciels, comme par exemple les serveurs Sunsite. Leurs données sont répliquées sur plusieurs *miroirs* dans le monde ; on se connecte au miroir le plus proche.

WAIS (*Wide Area Information Service*) est un service d'information par indexation de documents, très utile pour les recherches bibliographiques.

Dans une grande mesure, le Web a unifié ces différents services d'information distribuée, par l'adoption d'une interface unique à ces services.

Le Web a ses propres annuaires, qui sont de deux sortes :

- des répertoires, qui classifient les sites Web par catégories (par exemple Yahoo!, Infoseek)
- des moteurs de recherche, ou index, qui peuvent être interrogés par mots-clés (par exemple Alta Vista, HotBot, Google).

Les serveurs Web ne sont généralement pas répliqués, contrairement aux serveurs FTP (qui ont souvent plusieurs miroirs). L'accès aux données d'un serveur est facilité par la mise en place de caches qui conservent en mémoire les documents demandés, afin d'accélerer les transferts ultérieurs. Si un document se trouve dans un cache, il est transféré à partir de ce cache, et non à partir du serveur spécifié par la requête. Les caches sont organisé en une hiérarchie, qui va du cache de l'utilisateur (géré par le navigateur), le plus petit mais le plus rapide, au cache de l'opérateur de réseau, qui a une capacité bien plus grande. Il y a des niveaux de cache intermédiaire, notamment au niveau du site (par exemple, l'École des ponts).

1.4 Domaines

L'Internet étant une interconnexion de réseaux, toute machine raccordée à l'Internet est d'abord sur un réseau. A ce titre, elle est désignée par une *adresse IP*, qui spécifie le réseau, et à l'intérieur de ce réseau, la machine.

Une adresse IP se compose actuellement de 4 octets ; elle se décompose en numéro de réseau et numéro de machine. L'adresse du serveur Web de l'École est `195.221.195.17`, c'est la machine `17` dans le réseau `195.221.195`.

Plutôt que d'utiliser directement ces adresses IP, il est préférable de désigner les machines par des noms, plus faciles à retenir, et d'organiser ces noms de façon plus signifiante pour l'utilisateur. Par l'exemple, le nom www.enpc.fr indique immédiatement que www est dans le domaine enpc de l'Ecole, qui est dans le domaine français fr. L'association d'une adresse IP à un nom de machine est obtenue grâce au *service des noms*.

On a donc adopté rapidement un système de noms organisé hiérarchiquement en domaines. Au début, le système américain était composé de quelques domaines à trois lettres, au premier niveau de la hiérarchie : edu (établissements d'éducation), com (entreprises), gov (gouvernement), mil (armée), net (exploitation du réseau), org (organismes (n'appartenant pas aux autres types).

Depuis l'extension de l'Internet au reste du monde, à chaque pays correspond un domaine de premier niveau. Par exemple fi (Finlande), fr (France).

Chaque domaine peut avoir des sous-domaines. Par exemple enpc.fr est un sous-domaine de fr.

Le DNS est le *Domain Name System*. Il a la structure d'un arbre, dont tous les nœuds, sauf la racine, sont étiquetés par des noms.

A tout nœud de l'arbre correspond un domaine constitué de ce nœud et de ses descendants, et identifié par un nom de domaine unique, obtenu par concaténation des étiquettes des nœuds sur le chemin issu de la racine.

L'arbre des domaines est construit par un système d'autorités. Chaque autorité gère un ensemble de domaines à l'intérieur desquels elle a la possibilité de créer des sous-domaines et de déléguer leur gestion à d'autres autorités.

Une zone est constituée des sous-domaines d'un domaine gérés par une même autorité. Chaque zone est dotée d'une base de données maintenue par son autorité.

Le DNS utilise un système distribué de *serveurs de noms*, ce qui évite à chaque machine d'avoir une table complète de tous les noms et de leurs adresses IP. Si le serveur n'a pas la traduction dans sa table, il peut faire suivre la requête à un autre serveur ou bien envoyer au client l'adresse d'un autre serveur.

Ce sont souvent les mêmes requêtes qui sont envoyées, puisqu'une machine communique généralement avec un petit nombre d'autres machines ; et ce sont généralement les mêmes réponses qui sont reçues, car l'adresse IP d'une machine change rarement. Il est donc plus efficace de conserver en mémoire les réponses obtenues d'un serveur de nom. C'est le principe du *cache*, qui s'avère très utile en informatique.

Les réponses données par un serveur à une requête portant sur sa zone sont dites *d'autorité* (`authoritative answer`) et sont sûres ; les autres, qui proviennent d'une mise en mémoire, peuvent ne plus être correctes (ce qui est très rare). Les réponses ne sont conservées en mémoire que pendant un certain temps, spécifié dans les bases DNS, puis sont effacées afin de limiter le risque de retransmettre une valeur obsolète.

1.5 Protocoles et organisation en couches

Les protocoles et les services qu'ils offrent sont organisés en couches. L'idée est qu'une couche utilise des services fournis par la couche inférieure et offre des services à la couche supérieure. On distingue, par niveaux décroissants, les services d'application (les seuls qui aient un sens pour l'utilisateur), les services de transport, les services d'interconnexion des réseaux, et les services de liaison de données. Voici une « pile » de protocoles de l'Internet.

couche	*protocoles*
Application	Telnet, FTP, NNTP, SMTP, HTTP
Transport	TCP, UDP
Réseau	IP (ICMP)
Liaison	Ethernet, ...

Il existe d'autres piles de protocoles, par exemple AppleTalk.

couche	*protocoles*
Application	ASP, AFP
Transport	ADSP, ADP
Réseau	DDP
Liaison	LLAP

1.6 Ports et services de transport

Sur une machine, il peut y avoir plusieurs serveurs coexistants (par exemple un serveur FTP et un serveur HTTP). Il faut donc dire quel serveur doit traiter la requête émise par le client. On utilise pour cela un *numéro de port* attribué par les instances de l'Internet, connu de tous : 23 pour Telnet, 21 pour FTP, 80 pour HTTP, etc.

Il peut aussi y avoir plusieurs clients sur une machine, et même plusieurs clients requérant le même service. Il faut donc que le serveur puisse distinguer entre ces différents clients pour leur répondre correctement. On utilise aussi un numéro de port côté client, mais cette fois-ci le numéro de port est généré de manière à assurer l'unicité.

Une requête de transport sera donc exactement caractérisée par l'adresse IP du client, le numéro de port (unique) du client, l'adresse IP du serveur, le numéro de port (connu) du serveur. Ce sera un quadruplet, par exemple

(`192.54.211.35`, `6492`, `192.54.211.74`, `21`) dans le cas d'une connexion Telnet. On retrouve ces numéros de ports sur le Web dans certaines URL (par exemple `www.truc.fr:80`).

Un *protocole de transport* permet de réaliser un service d'application reliant deux processus, l'un serveur et l'autre client.

Un *service de transport* supporte un flot bidirectionnel de données entre les processus client et serveur, et est réalisé à l'initiative du client.

Une communication se fait en *mode connecté* s'il y a accord préalable entre l'émetteur et le récepteur sur l'établissement de la communication, avant que l'émission de données ne commence. Elle est en mode *non-connecté* sinon. La téléphonie fonctionne en mode connecté, la messagerie électronique en mode non-connecté.

Le mode connecté assure une meilleure sécurité à l'acheminement, et permet de *négocier la qualité du service* (par exemple, pour déterminer un taux de compression). Mais la mise en place d'une connexion demande toujours un temps supplémentaire, ce qui peut être pénalisant si le message transmis est court. Le mode non connecté est plus rapide et moins sûr.

Il y a deux protocoles de transport dans l'Internet :

- TCP (*Transmission Control Protocol*) offre un service de transport fiable en mode connecté
- UDP (*User Datagram Protocol*) offre un service de transport non fiable en mode non connecté

L'en-tête d'un datagramme TCP se compose de plusieurs champs, dont les plus significatifs sont les ports d'origine et de destination, un numéro de série qui permet de réordonner les datagrammes d'un même connexion et d'éliminer les duplicats, et un accusé de réception qui doit être renvoyé à l'expéditeur par le destinataire.

Le protocole UDP est beaucoup plus simple, ne permettant pas de reconstituer l'ordre d'envoi des messages, donc plus efficace, mais moins fiable, puisqu'il n'est pas doté d'accusé de réception.

1.7 Réseaux et modes de commutation

Il y avait jusqu'à présent deux grandes catégories de réseaux : les réseaux de télécommunications, basés sur la commutation de circuits, et les réseaux informatiques, basés sur la commutation de paquets. Ces deux catégories sont appelées à fusionner à moyen terme.

Les *réseaux de télécommunications* sont conçus pour *réaliser des services*, dont le principal est la téléphonie. Ces services ont généralement des contraintes temps réel (ou isochrones): l'acheminement de la voix doit être inférieur à 28 ms. Ces réseaux ont jusqu'à présent privilégié la *commutation de* *circuits* : un circuit physique est établi et utilisé pendant la durée de la communication entre l'émetteur et le récepteur. Le circuit établi suit une route dont les nœuds sont des autocommutateurs. Cela satisfait les contraintes isochrones (pour des réseaux terrestres d'échelle moyenne). Cependant quand les deux correspondants n'émettent rien, le circuit existe toujours, mais est inutilisé. Autre inconvénient, le réseau a une capacité maximale, ce qui peut conduire à des attentes pour l'établissement d'un circuit.

Un *réseau informatique* est un élément d'un système informatique permettant le *transport de données*. Il y en a de très petite taille : les bus reliant unités de calcul et mémoire d'un ordinateur.

Les réseaux informatiques peuvent être classés selon leur extension géographique :

- les réseaux locaux (LAN ou Local Area Network) sur quelques centaines de mètres (réseaux Ethernet, Token Rink, Apple Talk, etc)
- les réseaux métropolitains (MAN ou Metropolitan Area Network) jusqu'à quelques dizaines de kilomètres
- les réseaux régionaux, ou étendus (WAN ou Wide Area Network), au delà des réseaux métropolitains

Après les LAN installés dans les universités et les entreprises dès le début des années 80, après l'interconnexion des WAN pour constituer l'Internet dans les années 80, on assiste maintenant au développement des MAN qui bénéficieront des nouvelles technologies (ATM).

Les réseaux informatiques, qui n'avaient pas jusqu'à présent de contraintes isochrones, ont choisi un autre mode de commutation que celle des circuits. Une première idée consiste à réaliser une commutation basée sur les messages (entre les messages, il n'y a que du silence).

Comme il n'y a pas de circuit établi de bout en bout, le message est envoyé de nœud en nœud, et ne peut être envoyé au nœud B par le nœud A tant qu'il n'a pas été complètement et correctement reçu par A. Chaque nœud doit donc disposer d'une mémoire pour stocker les messages tant qu'ils ne sont pas stockés dans le nœud suivant.

Il est clair que si ce mode de commutation est adopté avec des messages de taille quelconque, plusieurs problèmes se posent :

- la taille de la mémoire de chaque nœud
- la probabilité qu'un message soit erroné, qui croît avec la taille du message

D'où l'idée de découper un message en paquets, de taille bornée (typiquement 1 koctet), et de faire de la commutation de paquets. Non seulement cela résoud les deux problèmes précédents, mais la durée totale d'acheminement du message se trouve réduite, puisqu'un nœud peut commencer à renvoyer un paquet dès qu'il l'a complétement reçu, sans attendre l'intégralité du message. Ce mode de commutation, dit *de paquets*, est celui d'Internet depuis ses débuts.

L'étape suivante consiste à passer de paquets de taille bornée, mais encore variable, à des "cellules" de taille fixe et très petite. Cela permet encore de réduire la durée d'acheminement, mais surtout d'augmenter la vitesse de commutation grâce à la réalisation de commutateurs matériels. C'est le choix de la technologie ATM, qui doit servir de base au futur RNIS large bande.

1.8 Le protocole IP

Le protocole IP permet l'acheminement de données entre une machine source et une machine destinataire, à travers un système de réseaux interconnectés. Il implémente deux fonctions essentielles : l'adressage et la fragmentation. L'acheminement des paquets est en mode non-connecté : les paquets sont indépendants les uns des autres, il n'y a pas de séquencement.

Un paquet[3] IP est composé d'un en-tête et de données (qui sont souvent elles-mêmes un datagramme de transport).

L'en-tête IP contient un certain nombre de champs, dont le protocole du service de transport utilisé pour les données du paquet (en général, TCP ou UDP), l'adresse IP d'origine et l'adresse IP de destination

Le protocole IP utilise ICMP (*Internet Control Message Protocol*) pour signaler des erreurs (par exemple : machine ou réseau inaccessible).

IP version 6 (ou IPng) est la nouvelle version du protocole IP qui sera progressivement mise en place pour corriger les défauts de la version actuelle.

1.9 L'Internet

En 1969, le département des recherches avancées de l'armée américaine (ARPA, maintenant DARPA) décidait de relier entre eux les ordinateurs de ses centres de recherche par une nouvelle technologie de transmission de données par paquets : l'Arpanet.

Le projet Internet date de quelques années plus tard, quand Vint Cerf et Bob Kahn imaginent une architecture unique qui permettait de fédérer les réseaux entre eux. Les premières versions des protocoles IP sont publiées en 1978, et mises en place en 1981. L'Internet résulte de l'interconnexion de réseaux communiquant par les mêmes protocoles de manière à offrir les mêmes services. Internet n'est donc pas *un* réseau.

Vers 1984, l'Internet est accessible depuis la France grâce à des passerelles entre l'Internet américain et le réseau français basé sur le service Transpac des télécoms ; le raccordement à l'Internet par une liaison directe INRIA–NSF est effectué en 1988. L'Internet commence à toucher le grand public en 1994, avec l'apparition du Web.

Il n'y a pas d'organisation centrale, que ce soit au niveau de l'exploitation technique, de l'administration du réseau, du contrôle de l'utilisation, ou de

3. Les RFC utilisaient le mot *datagram* pour désigner les paquets IP

la facturation des services. Il n'existe que des *instances techniques* chargées de l'élaboration et de l'adoption des protocoles, et de façon décentralisée, de l'attribution des adresses IP et des noms de domaine. Ces instances sont supervisées par l'*Internet Architecture Board* (IAB). Elles comprennent :

- l'IRTF (*Internet Research Task Force*) chargé de la recherche et de l'évolution à long terme de l'Internet
- l'IETF (*Internet Engineering Task Force*) chargé de l'élaboration des documents techniques (normes), au sein de groupes de travail spécialisés
- l'IESG (*Internet Engineering Steering Group*) chargé du pilotage des activités de l'IETF, et de l'adoption des standards
- l'IANA, chargé de l'attribution de certains numéros (en particulier, les ports)

L'IAB intervient comme instance d'appel de l'IESG pour l'adoption des normes.

L'activité de normalisation de l'Internet a complètement échappé aux organismes de normalisation, internationaux (l'ISO, l'UIT-T pour les télécoms) ou nationaux (ANSI aux USA, AFNOR en France, DIN en Allemagne, etc) qui sont composés de représentants d'organismes techniques dépendants des gouvernements.

L'Internet a conçu sa propre procédure de normalisation. N'importe qui peut être à l'origine d'une norme ; il suffit d'envoyer une proposition à l'IAB par courrier électronique. Les projets de normes sont rédigés par des groupes de travail, au sein de l'IETF, et rendus publics d'abord sous forme de *Internet Draft*, puis sous la forme de RFC (*Request For Comments*), avec l'état de *Proposed Standard Protocol*. Après une période de commentaires et de tests, il passe à l'état de *Draft Standard Protocol*, puis de *Standard Protocol*, ou bien de *Experimental Protocol* si les tests de validation ou les avis ne sont pas positifs.

Certains RFC ne définissent pas des normes, mais servent à la diffusion d'informations, et sont étiquetés FYI (*For Your Information*).

Tous les RFC (et même les Drafts Internet) sont disponibles via l'Internet sur de nombreux sites, notamment en France à l'adresse `<ftp://ftp.inria.fr/rfc/>`.

Chapitre 2

Documents hypermédias

2.1 La structure d'une page HTML

Le Web utilise HTML (*Hyper Text Marking Language*) comme langage de construction de documents hypermédias. C'est un langage de marquage : un fichier HTML contient à la fois les données du document et des instructions de mise en page logique appelées *marquages.* Il ne s'agit pas d'un langage de mise en page (comme Postscript). Les marquages décrivent seulement la structure logique du document, qui pourra être visualisé ou imprimé de façon diverse par le navigateur.

Chaque document HTML est défini par un fichier textuel (en ascii) dont le nom a `.html` (ou `.htm` pour certains systèmes) comme suffixe.

Les marquages apparaissent entre `<` et `>`. Ils ont à la fois une forme *ouvrante* et une forme *fermante* comme `<H1>` et `</H1>` : la forme fermante commence toujours par `</` (sauf pour les commentaires). Pour certains marquages, la forme fermante n'est pas indispensable. Les marquages ont souvent des *attributs*, éventuellement avec une valeur, qui sont toujours spécifiés dans la forme ouvrante ; par exemple, `<IMG SRC="toto.gif">` spécifie l'attribut `SRC`, avec la valeur `"toto.gif"`, pour le marquage `<IMG>`. Certains attributs sont obligatoires (c'est le cas de `SRC`, puisqu'une image est toujours spécifiée par un fichier), tandis que d'autres sont optionnels. Certains d'entre eux seront prochainement rendus obsolètes par l'usage des feuilles de style.

Les marquages peuvent indifféremment être écrits en majuscules ou en minuscules. Les blancs (entre les mots) et les sauts de ligne ne sont pas significatifs, sauf dans une région préformatée (et dans une ancre, mais de façon mineure).

Une page est composée d'un en-tête et d'un corps. Seul ce qui figure dans le corps est visible dans le document.

```
<HTML>
<HEAD>
<TITLE>Cours HTML</TITLE>
```

```
<!-- Auteur : R. Lalement -->
</HEAD>

<BODY>
   ....
</BODY>
</HTML>
```

L'en-tête contient des informations sur le document. Dans l'en-tête, le marquage <TITLE> ... </TITLE> définit le titre du document qui apparaît comme titre de la fenêtre du navigateur. Peuvent également y figurer des informations méta-textuelles dans le marquage <META>.

Les commentaires figurent entre <!-- et -->.

Certains caractères qui ont une signification spéciale ne peuvent pas être employés. Il s'agit d'abord de < et >, qui sont représentés par < et > ; on écrira par exemple 0 < 1. Comme cette représentation donne une signification spéciale à &, on représente ce symbole avec &.

Il existe plusieurs niveaux du langage HTML, de 1 à 4. Alors que tous les navigateurs comprennent HTML 2, seuls certains ont implémenté HTML 3. Le langage reconnu par Netscape est voisin de HTML 3 et comprend en outre des extensions qu'il est le seul navigateur à reconnaître et dont il vaut mieux ne pas abuser.

2.2 Le texte

HTML propose 6 niveaux de titres de sections, de <H1> ... </H1>, le plus grand, à <H6> ... </H6>, le plus petit. Ces titres sont affichés plus ou moins gros, sans numérotation.

Les paragraphes sont délimités par le marquage <P> ... </P>, qui admet les attributs suivants :

- ALIGN, avec les valeurs *center*, *left* ou *right*, aligne un paragraphe
- CLEAR, avec les valeurs *all*, *left*, *right*, permet d'aller à la ligne en présence d'images flottantes (voir 2.3).

On écrira par exemple un paragraphe justifié à droite :

```
<P ALIGN=right>
On écrira par exemple un paragraphe justifié à droite :
</P>
```

Il est aussi possible de centrer un paragraphe (ou une image, ou un tableau) avec le marquage <CENTER> ... </CENTER>, qui est propre à Netscape.

Le passage à la ligne dans un même paragraphe se fait par le séparateur
, qui peut avoir l'attribut CLEAR, comme pour <P>.

Le marquage <BLOCKQUOTE> ... </BLOCKQUOTE> sert aux citations, placées entre deux marges à droite et à gauche.

On obtient une ligne horizontale avec le marquage <HR> ; l'attribut SIZE spécifie sa hauteur (en pixels), l'attribut WIDTH spécifie sa largeur (en pixels ou en pourcentage de la largeur de la fenêtre du navigateur) :

```
<HR SIZE=5 WIDTH=75%>
```

Il existe plusieurs types de listes.

- les listes numérotées, par le marquage <OL>
- les listes non-numérotées, par le marquage <UL>
- les listes de définitions, par le marquage <DL>

Pour les deux premiers types de liste, chaque item doit être précédé de <LI>. Par exemple

```
<OL>
 <LI> Premier item
 <LI> Deuxième item
 <LI> Troisième item
</OL>
```

Pour les listes de définitions, utilisables par exemple pour mettre en forme un glossaire, chaque mot à définir est précédé de <DT> et sa définition est précédée de <DD> :

```
<DL>
 <DT> HTML
 <DD> Hyper Text Mark-up Language
 <DT> URL
 <DD> Uniform Resource Locator
</DL>
```

Le marquage <PRE> ... </PRE> définit un texte préformaté au sens où les blancs et sauts de lignes sont respectés. Il était souvent utilisé en HTML 2, quand les tableaux n'existaient pas encore. Il reste utile pour présenter des programmes.

2.3 Les images incrustées

Les images apparaissant dans un document doivent être en format GIF, JPEG ou PNG. Typiquement, le format JPEG (compression avec pertes) convient plutôt aux photographies, le format GIF (compression sans perte) aux dessins « au trait » ou aux logos (en particulier pour les images à fond transparent). Le format PNG est un nouveau format développé pour le Web mais encore peu utilisé.

Pour incruster une image, on utilise le marquage <IMG>, avec les attributs suivants

- SRC ; la valeur est l'*URL* de l'image (obligatoire)

- `ALT` ; la valeur est un texte qui sera affiché si l'image n'est pas demandée (la présence de cet attribut est fortement recommandée pour l'information de l'utilisateur)
- `WIDTH`, `HEIGHT` ; la valeur est un nombre entier de pixels, ou un pourcentage des dimensions de la page (la présence de cet attribut est fortement recommandée pour une meilleure mise en page)
- `BORDER` ; la valeur est un nombre entier de pixels
- `ALIGN` ; la valeur est *bottom*, *middle*, *top* (pour le positionnement vertical), *left* et *right* pour le positionnement horizontal permettant au texte de couler autour d'une image flottante
- `LOWSRC` (non-standard) ; la valeur est l'*URL* d'une image qui sera affichée par le client sur un écran à basse résolution, en remplacement de l'image spécifiée par `SRC`

Par exemple :

```
<IMG SRC="http://cermics.enpc.fr/images/logo.gif" ALT="logo"
     WIDTH=45 HEIGHT=50 ALIGN=middle>
```

2.4 Les ancres

A l'intérieur d'une page HTML, une référence à un autre document se fait en posant une *ancre* autour d'un texte ou autour d'une image : le texte sera souligné, l'image sera encadrée et il suffit de cliquer sur le texte ou l'image pour obtenir le document référencé.

Le marquage d'une ancre se fait par `<A>` ... `</A>`. L'URL d'un document référencé (ou pointé) est spécifié par l'attribut `HREF`. Autour d'un texte

```
Le <A HREF="http://cermics.enpc.fr/">CERMICS</A>.
```

et autour d'une image

```
<A HREF="http://cermics.enpc.fr/">
<IMG SRC="http://cermics.enpc.fr/images/logo.gif" ALT="logo">
</A>.
```

Il est également possible de pointer vers un point d'une page HTML. Ce point est spécifié par l'attribut `NAME` qui définit un *nom*. Par exemple,

```
en posant une <A NAME="ancre">ancre</A> autour d'un texte
```

La référence à ce point se fait en faisant précéder ce nom par `#` :

```
Voir la définition des <A HREF="#ancre">ancres</A>.
```

2.5 Les schémas d'accès

Différents schémas peuvent être utilisés pour accéder à un document ou plus généralement à une ressource, en recourant à certains protocoles de l'Internet.

HTTP (*HyperText Transfer Protocol*) est le protocole de base pour les documents situés dans un répertoire WWW :

```
<A HREF="http://www.enpc.fr/~moi/fichier.html"></A>
```

C'est aussi le protocole utilisé si une URL relative est indiquée :

```
<A HREF="/~lui/fichier.html"></A>
```

FTP (*File Transfer Protocol*) ouvre une session sur une machine distante pour le transfert de fichiers.

```
<A HREF="ftp://ftp.enpc.fr/pub/">Serveur FTP</A>
```

Il est possible d'indiquer le nom d'utilisateur qui doit être employé (ici, `login`) et le mot de passe (`passwd`)

```
<A HREF="ftp://login:passwd@ftp.enpc.fr/pub/">Serveur FTP</A>
```

Telnet ouvre une session interactive sur une machine distante. Le nom de la machine doit être spécifié ; il est aussi possible d'indiquer le nom d'utilisateur qui doit être employé (ici, `consult`).

```
<A HREF="telnet://consult@platon.enpc.fr">Platon</A>
```

Mailto envoie un message électronique à partir d'une page HTML :

```
<A HREF="mailto:Rene.Lalement@cermics.enpc.fr">René Lalement</A>
```

Le sujet du message peut être spécifié :

```
<A HREF="mailto:Rene.Lalement@cermics.enpc.fr?Subject:Info">René Lalement</A>
```

File ouvre un document local (qui ne se trouve pas nécessairement dans la racine WWW du serveur ni dans celle de l'utilisateur) ; le chemin d'accès absolu au fichier doit être donné :

```
<A HREF="file:/usr2/etudiants/moi/Web/un_fichier.html"></A>
```

Cependant ce document ne pourra pas être obtenu d'une machine distante.

2.6 Variations

Le langage HTML est beaucoup moins développé que d'autres langages pour le formattage de textes (notamment LaTeX). Il permet cependant quelques variations sur l'apparence des pages. Certaines possibilitées ne sont cependant reconnues que de Netscape. L'usage récent des feuilles de style permet d'uniformiser l'apparence des pages d'un site.

- `<EM>` ...`</EM>` : fonte italique
- `<STRONG>` ...`</STRONG>` : fonte grasse
- `<CODE>` ...`</CODE>` : fonte non-proportionnelle (ou à chasse fixe)

Il existe aussi des marquages `<I>`, `<B>`, `<TT>` qui donnent généralement les mêmes résultats, mais qu'il vaut mieux ne pas utiliser.

La taille et la couleur, pour Netscape, peuvent être modifiées localement par le marquage `<FONT>` et les attributs

- `SIZE`, avec une valeur allant de -7 à +7
- `COLOR`

Les couleurs du fond, du texte, des ancres et des ancres déjà visitées sont définissables par des attributs du marquage `<BODY>` : respectivement `BGCOLOR`, `TEXT`, `LINK`, `VLINK`. La couleur est spécifiée par `#xxyyzz`, où `xx`, `yy` et `zz` sont les codes hexadécimaux de ses composantes RGB. Par exemple :

- Blanc = `#FFFFFF`
- Noir = `#000000`
- Rouge = `#FF0000`
- Vert = `#00FF00`
- Jaune = `#FFFF00`
- Dark Purple = `#871F78`

Par exemple un gros titre dont la première lettre est plus grande que le reste et rouge peut être composé par :

```
<FONT SIZE=+5 COLOR=#FF0000>U</FONT><FONT SIZE=+3>n titre</FONT>
```

Voici un exemple :

```
<BODY BGCOLOR=#FFC573 TEXT=#000000 LINK=#CC3299 VLINK=#20A050>
```

Le fond peut aussi être un motif reproduit sur toute l'étendue de la page. Il suffit de spécifier un fichier GIF : `<BODY BACKGROUND = "`*motif*`.gif">`.

2.7 Les tableaux

Un tableau est construit par `<TABLE>` ... `</TABLE>`. L'attribut `BORDER` spécifie la taille de la bordure du tableau. Les attributs `CELLSPACING` et `CELLPADDING` spécifient la distance entre deux cases et respectivement, la distance entre le contenu et la bordure de chaque case.

Chaque ligne est entre `<TR>` ... `</TR>` ; chaque case est entre `<TD>` ... `</TD>`. Un premier exemple de tableau régulier :

```
<TABLE BORDER=4 CELLSPACING=4 CELLPADDING=3>
<TR><TD>1</TD><TD>2</TD><TD>3</TD></TR>
<TR><TD>4</TD><TD>5</TD><TD>6</TD></TR>
<TR><TD>7</TD><TD>8</TD><TD>9</TD></TR>
</TABLE>
```

Un tableau régulier sera mis en page plus rapidement si le nombre de colonnes est spécifié comme attribut de `<TABLE>` :

```
<TABLE BORDER=4 CELLSPACING=4 CELLPADDING=3 COLS=3>
```

A l'intérieur de `<TR>` et de `<TD>`, les attributs `ALIGN` et `VALIGN` positionnent le contenu d'une case, avec les valeurs *left*, *center*, *right* pour le positionnement horizontal et *top*, *middle* et *bottom* pour le positionnement vertical.

A l'intérieur de `<TD>`, l'attribut `ROWSPAN` spécifie le nombre de lignes sur lequel s'étend la case (par défaut, 1) ; l'attribut `COLSPAN` spécifie le nombre de colonnes sur lequel s'étend la case (par défaut, 1).

Voici un exemple avec les attributs :

```
<TABLE BORDER=4 CELLSPACING=4 CELLPADDING=3>
<TR><TD COLSPAN=2>AAAAAAAAAAAAAA</TD></TR>
<TR><TD ROWSPAN=2 VALIGN=middle ALIGN=center>BBBBBBBB</TD>
    <TD ALIGN=right>123456</TD>
</TR>
<TR><TD ALIGN=right>456</TD></TR>
</TABLE>
```

2.8 Les informations méta-textuelles

Dans l'en-tête du document, le marquage `<META> ... </META>` permet de spécifier certaines informations sur le document : par exemple, son auteur, une description, des mots-clés.

```
<META NAME="author" CONTENT="René Lalement">
<META
  NAME="description"
  CONTENT="Cours Web de l'ENPC">
<META
  NAME="keywords"
  CONTENT="HTML, Web, HTTP, CGI, JavaScript">
```

Ce marquage permet également de spécifier certains champs de l'en-tête de la réponse HTTP envoyée par le serveur au client, par exemple pour la date de péremption d'un document :

```
<META
  HTTP-EQUIV="Expires"
  CONTENT="Tue, 20 Aug 1996 14:25:27 GMT">
```

ou pour un rechargement automatique dans 2 minutes :

```
<META HTTP-EQUIV="Refresh" CONTENT="120">
```

On utilise aussi pour envoyer automatiquement une autre page, ce qui est commode quand un document a changé de nom ou d'adresse (respecter les guillemets simples et doubles) :

```
<HEAD>
<META HTTP-EQUIV="Refresh" CONTENT="5;URL='new.html'">
</HEAD>
```

2.9 Feuilles de style

L'apparence d'un document peut être déterminée par des règles de style. Il existe plusieurs langages de définition de style, celui utilisé ici est appelé *Cascading style sheet* ; le type MIME de ces définitions est « `text/css` ».

2.9.1 Règles de style

Un style peut être défini localement, par exemple

```
<H1 STYLE = "border-style:simple; border-width:3pt;">Un titre</H1>
```

ou bien globalement, dans l'en-tête du document, et appliqué à tous les éléments concernés, par exemple

```
<HEAD>
<STYLE>
H1 {border-style:simple; border-width:3pt;}
</STYLE>
</HEAD>

<BODY>
<H1>Un titre</H1>
</BODY>
```

ou encore globalement mais appliqué seulement aux éléments dont la classe correspond au style défini, par exemple

```
<STYLE>
H1.encadre {border-style:simple; border-width:3pt;}
```

```
</STYLE>

<H1 CLASS="encadre">Un titre</H1>
```

Comme il existe plusieurs langages de définition de style, et bien que celui utilisé ici soit le langage compris par défaut, il est plus correct de le spécifier par l'attribut `TYPE` :

```
<STYLE TYPE="text/css">
 ...
</STYLE>
```

Certains navigateurs, qui ignorent ce qu'est une règle de style, les affichent littéralement. Pour éviter cet inconvénient, il est conseillé de placer ces règles à l'intérieur de commentaires HTML :

```
<STYLE TYPE="text/css">
<!--
 ...
-->
</STYLE>
```

Les définitions locales sont réservées à des usages occasionnels ; pour obtenir des documents homogènes, de façon analogue aux documents LaTeX, il est recommandé d'utiliser des définitions globales et de limiter leur application grâce à l'attribut `CLASS`. La valeur de cet attribut est un nom de classe choisi par l'auteur pour distinguer diverses utilisations d'un même élément ; par exemple, la classe d'un paragraphe pourra être `intro`, `ordinaire` ou `conclusion` ; la classe d'un texte en emphase pourra être `rouge`.

Un style est défini par une suite de règles ; chaque règle est formée de zéro, d'un ou de plusieurs sélecteurs et d'un ensemble de déclarations.

Un sélecteur peut être le nom d'un élément (`BODY`, `P`, `EM`,. . .), le nom d'un élément suffixé par un nom de classe (`P.intro`, `P.ordinaire`, `EM.rouge`,. . .), ou encore un pseudo-élément comme `A:link` ou `A:visited`. Un sélecteur peut aussi être contextuel : par exemple, `H1 EM` s'applique aux textes en emphase à l'intérieur d'un titre de premier niveau, et `OL LI B` s'applique aux textes en gras dans un item d'une liste numérotée.

Les déclarations sont placées entre accolades. Chacune associe une valeur à une propriété, par exemple `width: 50%;` et est terminée par un point-virgule.

La première des règles suivantes s'applique à tous les `H1` et aux `H2` de classe *encadre* seulement ; la seconde s'applique aux `H3` de classe *encadre*.

```
<STYLE>
H1, H2.encadre {
  border-style:inset;
  border-width:4pt;
}
```

```
H3.encadre {
  border-style:ridge;
  border-width:2pt;
}
</STYLE>

<H1>Un titre</H1>
<H2 CLASS="encadre">Un autre titre</H2>
```

Le marquage `<SPAN>` permet d'entourer une zone du document à laquelle est limitée l'application d'un style :

```
<SPAN CLASS="encadre">
<H2>Un titre encadré</H2> ...
</SPAN>
```

2.9.2 Héritage

Les règles qui s'appliquent à un même élément se combinent. De plus, quand un élément en contient un autre, les règles qui s'appliquent à l'élément conteneur peuvent être héritées par l'élément contenu. Par exemple, on a la suite d'éléments, chacun contenant le suivant : `BODY`, `UL`, `LI`, `EM`. Si une règle est définie au niveau de `UL` (liste non numérotée), elle s'appliquera aussi aux `LI` (items) qu'elle contient, et aux `EM` (textes en emphase) que ces items contiennent. Cette propriété d'héritage n'est cependant pas valable pour toutes les propriétés ; par exemple, les marges, le type ou la couleur du bord d'un élément de type bloc (comme `P` ou `H1`) ne sont pas hérités par ses sous-éléments.

2.9.3 Feuilles de style externes

Le recours aux styles permet de rendre un site homogène en définissant des règles de style qui sont partagés par l'ensemble des documents de ce site.

Pour cela, ces règles doivent figurer dans un fichier, sans être incluses dans un marquage `<STYLE>` ; soit `style.css` ce fichier.

Pour appliquer ces règles à un document, il suffit d'inclure dans l'en-tête de ce document le marquage suivant :

```
<LINK REL="stylesheet" TYPE="text/css" HREF="style.css">
```

2.10 Des documents multimédias

Le format des messages électroniques est défini par la RFC 822, uniquement pour des messages textuels en ASCII ; son succès a été tel qu'il s'est rapidement imposé au delà de l'Internet. Cependant, il n'envisage pas la possibilité de messages multimédias, ni même de jeux de caractères plus étendus que l'ASCII (comme les caractères accentués français). C'est pourquoi, les extensions *MIME* (Multipurpose Internet Mail Extensions) ont été introduites et servent plus généralement à spécifier la nature des informations circulant sur l'Internet et en particulier sur le WWW. [RFC 2045 à 2049] Tous les logiciels de messagerie ne supportent pas MIME ; Eudora (sur Mac ou PC) et Netscape (toutes plateformes) implémentent une grande partie des extensions MIME.

Le format MIME ajoute à l'en-tête habituel des messages électroniques plusieurs champs, notamment :

- `MIME-Version` indique que le document a été composé en conformité avec les extensions MIME, en spécifiant le numéro de version utilisée, qui est actuellement `1.0`
- `Content-Type` spécifie le type (et le sous-type) des données et le codage d'origine de ces données : les valeurs du type peuvent être `text`, `image`, `audio`, `video`, `message`, `application`, `multipart`
- `Content-Transfer-Encoding` spécifie le codage des données utilisé afin de permettre leur transport

Quelques type/sous-types courants :

- `text/plain` : document textuel non formatté
- `text/html` : document HTML
- `text/css` : feuille de style en format CSS
- `multipart/mixed` : document composé de plusieurs parties MIME, chacune ayant son propre `Content-Type`
- `image/jpeg` : image en format JPEG
- `image/gif` : image en format GIF
- `audio/basic`
- `video/mpeg`
- `video/quicktime`
- `application/octet-stream` : document provenant d'une application, qui doit simplement être interprété comme une suite d'octets
- `application/postscript` : document en Postscript
- `application/rtf` : document en RTF, format portable de Word
- `message/rfc822` : document composé d'un message électronique (utile pour faire suivre un message)

En plus des valeurs, les champs de l'en-tête peuvent définir des paramètres. Par exemple, `Content-type: text/plain; charset=us-ascii` indique que le jeu de caractères 7 bits américain, tandis que `Content-type: text/plain; charset=iso-8859-1` indique le jeu de caractères 8 bits utilisé par les langues européennes. Ce dernier, appelé généralement iso-latin1, comporte les caractères accentués majuscules ÀÁÂÃÄÅ, Æ, Ç, ÈÉÊË, ÌÍÎÏ, Ð, Ñ, ÒÓÔÕÖ, Œ, Ø, ÙÚÛÜ, et Ý, en majuscules et minuscules et d'autres symboles ; le caractère œ du français n'y figure cependant pas.

Le type `multipart` utilise un paramètre `boundary` qui spécifie la ligne délimitant les différentes parties MIME du document.

Le champ `Content-Transfer-Encoding` spécifie le codage utilisé pour transporter les données (un message électronique ne peut contenir que des caractères 7 bits, éventuellement 8 bits, et des lignes de moins de 1000 caractères). Dans le cas d'un type `multipart`, chaque partie a son propre champ `Content-Transfer-Encoding`. Les valeurs courantes de ce champ sont :

- `7bit` : caractères ASCII 7 bits (pas de codage)
- `8bit` : caractères ASCII 8 bits (pas de codage)
- `quoted-printable` : codage des octets qui ne sont pas des caractères 7 bits par '=' suivi de la représentation hexadécimale (à 2 chiffres) de la valeur de l'octet (par exemple `é` est codé par `=E9`) ; les fins de ligne sont codés par `=` en fin de ligne ; ce codage reste assez lisible
- `base64` : codage d'une suite quelconque d'octets (provenant par exemple d'un fichier binaire) par un sous-ensemble de 64 caractères ASCII et de `=` ; le principe est de grouper la suite d'octets par 6 bits et de coder chaque groupe par un caractère de ce sous-ensemble

Chapitre 3

Interactions

Le WWW n'est plus seulement un système d'information où l'unique opération est la navigation d'un document à l'autre, les seules requêtes émises par un client étant la demande d'un document, et la seule réponse du serveur l'envoi d'un document existant. Il permet d'autres *interactions* que le suivi d'un lien.

Le Web est désormais utilisé pour construire des interfaces graphiques universelles, accessibles à distance depuis n'importe quelle plateforme, pour des traitements de données qui peuvent être réalisées par le client ou par le serveur.

Ce chapitre décrit les éléments HTML qui permettent la construction de ces interfaces, tandis que les deux chapitres suivants développent les techniques de traitement, sur le serveur (scripts CGI), puis sur le client (scripts JavaScript).

3.1 Les formulaires

HTML permet de construire très facilement des interfaces graphiques à base de boutons, menus, listes de choix, etc, ces éléments étant regroupés dans un *formulaire*.

Un formulaire est défini par le marquage `<FORM>`, dont les attributs `ACTION` et `METHOD` spécifient le traitement des données saisies par l'utilisateur.

3.1.1 Les composants d'interface

Ces composants doivent figurer à l'intérieur du marquage `<FORM>` ... `</FORM>`.

Chaque composant a un type et un nom. Les noms des composants d'un même formulaire doivent être distincts, pour permettre le traitement des données saisies par l'utilisateur.

Ligne de texte

```
Prénom : <INPUT NAME="Prenom" TYPE="text" SIZE="10">
```

Zone de texte

```
Commentaires : <TEXTAREA NAME="Comm"  COLS="30" ROWS="5"></TEXTAREA>
```

Un texte par défaut peut être écrit entre les deux marquages :

```
Commentaires : <TEXTAREA NAME="Comm"  COLS="30" ROWS="5">
               Tout va bien.
               </TEXTAREA>
```

Ligne de texte invisible

```
Mot de passe : <INPUT NAME="Passwd" TYPE="password" SIZE="10">
```

Les caractères entrés seront affichés `*`, ce qui est commode pour entrer un mot de passe à l'abri des regards indiscrets.

Liste à choix multiple

```
<P> Langage(s) utilisé(s)
<INPUT NAME="lang" TYPE="checkbox" VALUE="HTML" CHECKED>
<INPUT NAME="lang" TYPE="checkbox" VALUE="C" >
<INPUT NAME="lang" TYPE="checkbox" VALUE="JavaScript" CHECKED>
```

Plusieurs boîtes pourront être cochées. L'option `CHECKED` permet de présélectionner certaines boîtes.

Liste à choix unique

```
<P> Sexe :
<INPUT NAME="Sex" TYPE="radio" VALUE="F">
<INPUT NAME="Sex" TYPE="radio" VALUE="M">
```

Au plus une boîte pourra être cochée.

Bouton

```
<P> <INPUT NAME="Fermer" TYPE="button" VALUE="fermer">
```

Bouton de ré-initialisation

```
<P> <INPUT TYPE="reset" VALUE="abandonner">
```

En pressant ce bouton, toutes les données déjà saisies seront effacées.

Bouton d'envoi

```
<P> <INPUT NAME="Envoyer" TYPE="submit" VALUE="envoyer">
```

En pressant ce bouton, les données saisies seront traitées, comme indiqué par l'`ACTION` du formulaire. Le nom de ce bouton n'est nécessaire que s'il y a plusieurs boutons d'envoi dans un formulaire, pour déclencher des traitements différents.

Menu à choix unique

Ces menus ont une fonction analogue aux listes à choix unique, mais une présentation différente.

```
<P> Sexe :
<SELECT NAME="Sex">
  <OPTION VALUE="F"> féminin
  <OPTION VALUE="M"> masculin
</SELECT>
```

L'attribut optionnel `SIZE` spécifie le nombre d'options qui seront affichées dans un tableau (sans `SIZE`, une seule option sera affichée) :

```
<P> Sexe :
<SELECT NAME="Sex" SIZE=2>
  <OPTION VALUE="F"> féminin
  <OPTION VALUE="M"> masculin
</SELECT>
```

L'attribut optionnel `SELECTED` permet de présélectionner une option par défaut.

Menu à choix multiple

Ces menus ont une fonction analogue aux listes à choix multiple, mais une présentation différente.

```
<P> Glace :
<SELECT NAME="Parfums" SIZE=2 MULTIPLE>
  <OPTION VALUE="C"> Chocolat
  <OPTION VALUE="F"> Fraise
  <OPTION VALUE="V" SELECTED> Vanille
</SELECT>
```

L'attribut optionnel `SELECTED` permet de présélectionner une ou plusieurs options.

3.1.2 Traitement de données

Toutes les données sont intégrées dans une *longue chaîne* de caractères, dont la forme générale est :

$$name_1\texttt{=}value_1\texttt{\&}name_2\texttt{=}value_2\texttt{\&}\ \ldots\texttt{\&}name_n\texttt{=}value_n$$

De plus, un espace est représenté par un `+`, et les caractères spéciaux par `%xx`, où `xx` est le code hexadécimal du caractère : par exemple, `<` par `%3C`, un `&` par `%26`.

C'est cette longue chaîne qui est transmise quand l'utilisateur presse le bouton d'envoi.

La valeur de l'attribut `ACTION` de `<FORM>` est une URL qui détermine l'action qui est alors exécutée, avec cette longue chaîne comme donnée.

Si les données peuvent être traitées localement, c'est-à-dire par le client, on utilisera un script local (JavaScript).

Si les données doivent être transmises au serveur, par exemple pour interroger une base de données accessible au serveur, on utilisera une action HTTP.

Action `mailto`

Cette action permet d'envoyer la chaîne des données par courrier électronique à une adresse spécifiée. Ceci ne provoquera aucun traitement automatique.

```
<FORM METHOD="post" ACTION="mailto:rl@enpc.fr">
```

Il est possible de spécifier certains champs de l'en-tête du message, par exemple :

```
<FORM
  METHOD="post"
  ACTION="mailto:rl@enpc.fr?Subject=info&Cc=keriven@enpc.fr">
```

Action `javascript`

Cette action permet d'exécuter des instructions JavaScript lors de la soumission des données.

```
<FORM ACTION="javascript:......">
```

Il est aussi possible d'exécuter des instructions JavaScript lorsque des événements sont déclenchés.

Action HTTP

Pour envoyer les données à un serveur HTTP (qui n'est pas nécessairement le serveur sur lequel se trouve le formulaire), il faut utiliser une action HTTP.

```
<FORM METHOD="post" ACTION="http://.....">
```

La valeur de l'attribut `ACTION` est l'URL d'un programme qui va traiter la chaîne des données. Ce programme reçoit ces données du client et envoie d'autres données au serveur HTTP qui va répondre au client : il réalise donc une interface client-serveur, appelée CGI (*Common Gateway Interface*).

3.2 Documents exécutables

Quand un navigateur (client) demande un document WWW ordinaire, celui-ci lui est envoyé par le serveur. Dans le cas d'un document exécutable, il s'ajoute une phase d'exécution. En général, un document exécutable est donc un programme (on dit aussi un *script*).

Il y a deux sortes de documents exécutables : du côté du serveur, du côté du client. Un *document exécutable du côté du serveur* est d'abord exécuté par le serveur, et c'est le résultat de l'exécution qui est renvoyé au client puis affiché par celui-ci. Un *document exécutable du côté du client* est d'abord envoyé au client, puis il est exécuté par le client, et c'est le résultat de cette exécution qui est affiché.

Alors qu'il n'existait, aux débuts du Web, que des documents exécutables du côté du serveur, on s'oriente de plus en plus vers des documents exécutables du côté du client grâce à des langages de script comme JavaScript.

3.3 Les images cliquables

On peut définir, à l'intérieur d'une image, certaines *zones cliquables* qui réfèrent à d'autres documents ; ce sont des documents exécutables, soit du côté du serveur (ce qui nécessite d'écrire des fichiers supplémentaires), soit du côté du client (ce qui est plus simple, mais que tous les navigateurs ne reconnaissent pas).

Côté client, on définit une *map* par le marquage `<MAP> ... </MAP>`, dont on spécifie le nom par l'attribut `NAME` ; à l'intérieur de ce marquage, on définit des zones par le marquage `<AREA>`, avec les attributs `SHAPE`, `COORDS` et `HREF`. Enfin, l'image est définie avec le nouvel attribut `USEMAP` ayant pour valeur le nom de la *map* qui a été définie, précédé d'un `#`.

```
<MAP NAME="carte">
<AREA SHAPE=rect COORDS="10,40, 100,200" HREF=/F1.html>
<AREA SHAPE=rect COORDS="120,5, 200,30" HREF=/F2.html>
```

```
<AREA SHAPE=default  HREF=/F0.html>
</MAP>
<IMG SRC="carte.gif" USEMAP="#carte">
```

3.4 Les panneaux

Quand le navigateur est lancé, il ouvre une fenêtre dans laquelle un document sera chargé. Ce document pourra diviser cette fenêtre en plusieurs sous-fenêtres, appelées panneaux, dont chacune pourra afficher à son tour un document. Cette division est réalisée par le marquage `FRAMESET` qui définit un *cadre* (ou jeu de panneaux), à partir de ses constituants qui sont, soit d'autres cadres, soit des *panneaux*.

3.4.1 Les cadres

Il est possible de partager un document, jusqu'ici défini par un unique `<BODY>`, en plusieurs parties, appelées *cadres* (anglais *frames*), définies par le marquage `<FRAMESET>` ... `</FRAMESET>`, et organisées comme un tableau.

Un fichier HTML contient soit un `<BODY>` qui définit le corps du document, soit un `<FRAMESET>` qui définit la géométrie du cadre et spécifie ses constituants (mais il ne doit pas y avoir à la fois `<BODY>` et `<FRAMESET>`). Un `<FRAMESET>` peut être inclu dans un autre `<FRAMESET>`.

```
<FRAMESET ROWS="20%,70%,10%">
```

définit trois lignes, qui occupent respectivement 20%, 70% et 10% de la hauteur de la fenêtre, de haut en bas.

```
<FRAMESET COLS="50,*">
```

définit deux colonnes, dont la première occupe 50 pixels et la seconde le reste de la largeur de la fenêtre, de gauche à droite.

`<FRAMESET>` a l'un des deux attributs `COLS` et `ROWS`, qui définissent la disposition des cadres contenus en lignes ou en colonnes. La valeur de ces attributs est une liste, généralement de pourcentages.

- `COLS=20%,50%,30%` définit trois colonnes qui occupent 20%, 50% et 30% de la largeur
- `COLS=300,*` définit deux colonnes, l'une de 300 pixels, l'autre occupant le reste de la la largeur
- `ROWS=300,2*,*, 100` définit quatre lignes, la première de 300 pixels de haut, la dernière de 100 pixels, le reste de la hauteur étant réparti aux seconde et troisième lignes dans la proportion de 2/3 et 1/3.

L'attribut `BORDER=0` permet de supprimer le bord du cadre.

Panneaux

A l'intérieur de `<FRAMESET>` ... `</FRAMESET>`, on trouve la définition des cadres spécifiés par le marquage `<FRAME>`. Ce marquage a un attribut essentiel `SRC` dont la valeur est l'URL du contenu du cadre. À chaque panneau, défini par le marquage `<FRAME>`, est associé un nom, qui le désigne, et un URL, qui spécifie son contenu.

```
<FRAME NAME="table" SRC="tdm.html">
```

D'autres attributs concernent la présentation du cadre :

- `MARGINWIDTH` et `MARGINHEIGHT` spécifient la dimension des marges horizontales et verticales du cadre, en pixels ;
- `SCROLLING`, avec pour valeurs `yes`, `non` ou `auto`, pour une barre de défilement (ascenseur ou tapis roulant)
- `NORESIZE` interdit au client de modifier la taille du cadre dans la page.

A l'intérieur de `<FRAMESET>` ... `</FRAMESET>`, le marquage `<NOFRAMES>` ... `</NOFRAMES>` sert à spécifier ce qui doit apparaître sur un navigateur qui n'affiche pas les cadres.

3.4.2 Ancres et panneaux

Le nom spécifié par l'attribut `NAME` d'un panneau sert à diriger l'affichage d'un document dans ce panneau, grâce à l'attribut `TARGET` des ancres.

Par exemple, la table des matières, dans le fichier `tdm.html`, contiendra des liens de la forme

```
<A HREF="chap2.html" TARGET="texte">Chapitre 2</A>
```

qui affichent les chapitres dans le panneau `texte`. Si un panneau n'est destiné à afficher qu'un seul document, il n'est pas nécessaire de lui attribuer un nom .

Pour afficher un document dans la fenêtre parente, le cadre précédent disparaissant, on utilise le nom `_parent` :

```
<A HREF="first_page.html" TARGET=_parent>Le Web</A>
```

Pour afficher en pleine page, tous les cadres disparaissant, on utilise le nom `_top` :

```
<A HREF="../home.html" TARGET=_top>Home page</A>
```

3.5 Les couches

HTML permet désormais de spécifier le positionnement des éléments d'un document dans la fenêtre du navigateur, par l'usage des règles de style. Ce positionnement peut être absolu ou relatif. Il est spécifié par les propriétés de style `position`, `top`, `left`, `z-order`. Un élément pour lequel la propriété `position` est définie est appelé une *couche* (anglais *layer*). Comme un positionnement ne servira en principe qu'à une seule occurrence d'un élément (et non, par exemple à tous les paragraphes d'un document), il est d'usage de regrouper les déclarations des propriétés d'une couche dans un style nommé, qui ne sera utilisé qu'une fois.

```
<STYLE>
#couche1 {
  position: absolute;
  top: 2cm;
  left: 2cm;
  width: 6cm;
  background-color: #FF0000;
}
</STYLE>
```

Le style nommé `couche1` sera appliqué ainsi :

```
<SPAN ID="couche1">
 ...
</SPAN>
```

Les éléments ayant un positionnement absolu sont représentés comme des boîtes rectangulaires à l'emplacement désigné. Si le positionnement de plusieurs couches entraîne leur recouvrement, on peut spécifier l'ordre de superposition à l'aide de la propriété `z-index` :

```
<STYLE>
#couche1 {
  position: absolute;
  top: 2cm;
  left: 2cm;
  width: 6cm;
  z-index: 1;
}
#couche2 {
  position: absolute;
  top: 3cm;
  left: 3cm;
  width: 6cm;
  z-index: 2
}
</STYLE>
```

Les éléments ayant un positionnement relatif sont représentés comme des boîtes rectangulaires placées relativement à la position courante (à la suite des éléments les précédant dans le corps du document).

3.6 Chaînes Web

Le marquage `<META>` de l'en-tête, avec l'attribut `HTTP-EQUIV` de valeur `Refresh` permet d'implémenter facilement un défilement de pages, simulant une chaîne sur le Web. La première page envoyée contient une temporisation (ici 3 secondes), et l'URL d'un document qui sera chargé à l'expiration de ce délai. Il est fortement déconseillé d'écrire des chaînes récursives.

```
<HEAD>
<META HTTP-EQUIV="Refresh" CONTENT="3;URL='page2.html'">
</HEAD>
<BODY>
Cette première page sera suivie dans 3 secondes d'une deuxième page.
</BODY>
</HTML>
```

L'idée d'organiser le Web comme une chaîne de télévision, très attirante pour ses applications commerciales, est en train de s'imposer.

Chapitre 4

Programmes CGI

4.1 CGI et HTTP

Le *Common Gateway Interface* spécifie l'interface entre les programmes exécutés par la machine serveur, le processus client (le navigateur) et le processus serveur HTTP. Les programmes dont les entrées et sorties vérifient cette spécification, que l'on appelle des *programmes CGI*, peuvent être écrits en n'importe quel langage : on utilisera ici C, alors que d'autres auteurs utilisent PERL, sh, C++ ou Java. CGI lui-même n'est pas un langage.

La compréhension de cette interface nécessite une connaissance minimale du protocole HTTP. Voici un extrait de la RFC 2068 qui définit la norme HTTP 1.1 :

> *The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible entity-body content.*

Parmi les différentes méthodes qu'une requête HTTP peut indiquer, les deux plus importantes sont `GET` et `POST`.

La méthode `GET` est celle que le navigateur forme quand l'utilisateur clique sur une ancre pour accéder à un document. Si ce document est un programme exécutable, c'est le résultat de l'exécution, écrit sur la sortie standard de ce programme qui sera transmis en réponse par le serveur au navigateur. Dans ce cas, l'URI du document peut être complétée par un « `?` » suivi d'une *longue chaîne* de même format que celles construites par le traitement des formulaires : cette chaîne se compose de couples *nom=valeur* séparés entre eux par &, sans aucun espace (un blanc dans une valeur est représenté par +) et les autres caractères non alphanumériques sont codés par % suivi du code hexadécimal de ce caractère. Voici un exemple d'une telle URI :

```
<A
HREF="/cgi-bin/calcul.cgi?val1=25&val2=39&texte=comment+%E7a+va%3F">
un programme</A>
```

Le programme accède à cette chaîne par l'intermédiaire de la variable d'environnement `QUERY_STRING` (voir ci-dessous). Les fonctions d'interrogation des annuaires du Web utilisent fréquemment cette méthode `GET`.

La méthode `POST` permet d'envoyer des données au serveur et de les faire traiter par un programme qui les lit sur son entrée standard. Comme pour `GET`, le résultat de l'exécution de ce programme écrit sur sa sortie standard sera transmis en réponse par le serveur au navigateur. Nous utiliserons uniquement cette méthode pour le traitement des formulaires.

4.2 Le traitement des formulaires

4.2.1 La longue chaîne

Considérons un document contenant le formulaire suivant, avec deux champs textuels simples et un bouton d'envoi :

```
<FORM METHOD="post" ACTION="cgi-bin/prog.cgi">
Lettre : <INPUT NAME="L" TYPE="text" SIZE="1"> <BR>
Chiffre : <INPUT NAME="C" TYPE="text" SIZE="1"> <BR>
<INPUT NAME="Envoyer" TYPE="submit" VALUE="envoyer">
</FORM>
```

l'utilisateur écrit `U` dans le premier champ, `2` dans le second, puis presse le bouton d'envoi.

Le navigateur construit alors la *longue chaîne* de caractères `L=U&C=2` et la transmet au serveur. Celui-ci déclenche l'exécution du programme `cgi-bin/prog.cgi` ; sur l'entrée standard de ce programme se trouve la chaîne `L=U&C=2` ; de plus, ce programme a accès à des variables d'environnement, notamment aux deux suivantes :

- la variable `CONTENT_LENGTH`, dont la valeur est la chaîne de caractères représentant la longueur de la chaîne transmise (c'est la chaîne `"7"` dans le cas de `"L=U&C=2"`)
- la variable `CONTENT_TYPE`, dont la valeur est la chaîne `"application/x-www-form-urlencoded"`

Note : les variables d'environnement Quand un utilisateur exécute un programme à partir du *shell*, un certain nombre de variables d'environnement sont accessibles ; leurs valeurs sont des chaînes de caractères. Les plus utiles sont `HOME`, dont la valeur est le répertoire personnel de l'utilisateur (ou *home directory*, `PATH`, pour la liste des répertoires où les commandes sont recherchées, `DISPLAY`, qui identifie l'écran sur lequel les fenêtres X Window s'affichent. On peut obtenir la liste de toutes ces variables par la commande `printenv`.

Un programme CGI ne sera pas exécuté à partir du shell, mais par le serveur ; c'est lui qui définit les variables d'environnement `CONTENT_LENGTH`, etc.

La fonction C, déclarée dans `stdlib.h`,

```
char *getenv(char *name);
```

permet d'obtenir la valeur de la variable d'environnement passée en argument.

4.2.2 La forme d'un programme CGI

Une application typique aura la forme suivante :

```
int main()
{
  couple couples[MAX];
  int m;
  char *longue_chaine = extraire_couples(couples, &m);

  decoder_couples(m, couples);
  creer_document(m, couples);
  enregistrer(FICHIER, m, couples);
  free(longue_chaine);
  return 0;
}
```

4.2.3 L'appel d'un programme CGI

Les programmes CGI figurent généralement comme valeurs de l'attribut `ACTION` d'un formulaire. Mais l'URL d'un programme CGI peut figurer là où un URL est demandée, notamment dans un lien.

Cela permet au même programme de *générer* un questionnaire et de le *traiter*.

4.2.4 Extraction des couples nom/valeur

On suppose que la méthode utilisée est `post`. Un programme robuste devrait commencer par tester qu'il s'agit bien de cette méthode, en comparant la valeur de la variable d'environnement `REQUEST_METHOD` (obtenue par un appel `getenv(REQUEST_METHOD)`) à la chaîne `POST` :

```
if(strcmp(getenv("REQUEST_METHOD"),"POST")) ...
```

La première étape consiste à extraire les couples nom/valeur de la longue chaîne ; on utilisera le type suivant :

```
typedef struct {
    char *nom;
    char *valeur;
} couple;

int main() {

  couple couples[MAX];
  int m;                    /* nombre de couples extraits */
  char *longue_chaine = extraire_couples(couples, &m);

  ...
}
```

On commence par allouer une chaîne de caractères sur le tas, dont la longueur est connue grâce à la variable d'environnement `CONTENT_LENGTH`. Puis on lit, dans cette chaîne, ce qui est envoyé sur l'entrée standard du programme, au moyen de

```
char *debut = gets(longue_chaine);
```

qui remplace le `\n` final par `\0`, le caractère nul. Les deux pointeurs `debut` et `longue_chaine` désignent maintenant le début de la longue chaîne.

On déplace deux pointeurs, `debut` et `fin`, dans `longue_chaine`. La fonction `char *index(char *s, char c)`, déclarée dans `strings.h` retourne un pointeur vers la première occurrence du caractère `c` dans la chaîne `s`, ou bien le pointeur `NULL` si `c` ne figure pas dans `s`. On isole successivement le premier nom, jusqu'à la première occurrence de `=`, puis la première valeur, jusqu'à la première occurrence de `&`, et ainsi de suite ; quand une occurrence de `=` ou `&` est trouvée, remplace ce caractère par `\0`, puis on fait pointer `debut` sur le caractère suivant `fin`, et on itère. On s'arrête quand on ne trouve plus d'occurrence de `&`. Le nombre de couples extraits est affecté à la variable `*m`, et la longue chaîne est retournée.

```
char *extraire_couples(couple couples[], int *m)

{
  int longueur = atoi(getenv("CONTENT_LENGTH"));
  char *longue_chaine = malloc((longueur + 1) * sizeof(char));
  char *debut, *fin;
  int i;
```

```
  debut = fin = gets(longue_chaine);
  for (i=0; fin != NULL && i < MAX; i++) {
    fin = index(debut, '=');
    if (fin != NULL) {
      couples[i].nom = debut;
      *fin = '\0';
      debut = fin + 1;
    } else break;
    fin = index(debut, '&');
      couples[i].valeur = debut;
    if (fin !=NULL) {
      *fin = '\0';
      debut = fin + 1;
    }
  }
  longue_chaine[longueur] = '\0';
  *m = i;
  return longue_chaine;
}
```

4.2.5 Le décodage

Dans les champs textuels d'un formulaire, un espace est codé par un `+`. De plus, les caractères spéciaux sont codés par `%xx`, où `xx` est le code hexadécimal du caractère : par exemple, un `<` par `%3C`, un `%` par `%25`.

La deuxième étape de traitement consiste donc à décoder les chaînes pour retrouver ces caractères. On utilisera par exemple les fonctions suivantes.

```
int hexa_vers_caractere(char s[2])
{
  int c;

  c = (s[0] >= 'A' ? s[0] - 'A' + 10 : s[0] - '0');
  c *= 16;
  c += (s[1] >= 'A' ? s[1]  - 'A' + 10 : s[1] - '0');
  return c;
}

void decoder_valeur(char s[])
{
    int i, j=0;

    for(i=0; s[j]!='\0'; i++) {
      switch (s[j]) {
      case '+':
        s[i] = ' ';
        j++;
        break;
```

```
      case '%':
        s[i] = hexa_vers_caractere(&s[j+1]);
        j += 3;
        break;
      default:
        s[i] = s[j];
        j++;
      }
    }
    s[i] = '\0';
}

void decoder_couples(int m, couple couples[])
{
  int i;

  for (i=0; i<m; i++) {
    decoder_valeur(couples[i].valeur);
  }
}
```

4.2.6 Enregistrer la requête dans un fichier

Pour une application telle qu'un vote, il sera nécessaire de conserver toutes les requêtes dans un même fichier.

```
#define FICHIER "/u/cermics/4/rl/www/cgi/resultats"


int main()
{
  ...
  enregistrer(FICHIER, m, couples);
  ...

}
```

Il est nécessaire de donner le chemin complet du fichier, et pas seulement son nom dans le répertoire courant.

```
void enregistrer(char nom_fichier[], int m, couple t[])
{
  int i;

  FILE *fp = fopen(nom_fichier, "a");

  if (!fp) {
    exit(1);
  }
```

```
  for(i=0; i < m-1; i++){
    fprintf(fp, "%s = %s, ",
            t[i].nom,
            t[i].valeur);
  }
  fprintf(fp, "%s = %s.\n",
          t[m-1].nom,
          t[m-1].valeur);
  fclose(fp);
}
```

Dans le cas où plusieurs requêtes peuvent être exécutées simultanément, il est préférable de poser un verrou sur le fichier, de sorte qu'un seul processus à la fois accède à ce fichier. Le verrouillage et le déverrouillage sont réalisés par la fonction `flock` déclarée dans `sys/file.h`.

4.3 Construction d'un document HTML envoyé au client

En général, le programme CGI doit construire un nouveau document qui va être renvoyé au client ; au minimum, ce sera un message de remerciement du genre « votre requête est enregistrée, merci », et souvent, le document retourné intégrera les données entrées par l'utilisateur dans un formulaire. C'est ainsi que l'on peut réaliser des sites interactifs.

Pour produire ce document, le programme CGI écrit sur sa sortie standard.

4.3.1 Document statique

S'il s'agit seulement d'envoyer un document statique, c'est-à-dire un fichier déjà existant, il suffira de préciser l'URL relative de ce fichier :

```
#include <stdio.h>

int main()
{
  printf("Location: contenu.html\n\n");
  return 0;
}
```

La ligne `Location:  contenu.html\n\n`, écrite sur la sortie standard du programme CGI contient une directive du protocole HTTP ; les deux `\n` sont obligatoires.

4.3.2 Document dynamique

Pour construire un document dynamique, c'est une autre directive HTTP qu'il faut utiliser, pour spécifier le type du document (ici, du HTML) au client.

```
#include <stdio.h>

int main()
{
  printf("Content-type: text/html\n\n");
  printf("<HTML>\n");
  ...
  printf("</HTML>\n");
  return 0;
}
```

Par exemple, la fonction suivante sert à construire un document dynamique à partir des données entrées par l'utilisateur dans un formulaire (à condition qu'elles ne comportent pas de caractères spéciaux pour HTML):

```
void afficher_couples(int m, couple t[])
{
  int i;

  printf("<H1>Résultats</H1>");
  printf("Les choix suivants ont été enregistrés :\n");
  printf("<UL>\n");
  for(i=0; i < m; i++)
    printf("<LI> <CODE>%s = %s</CODE>\n", t[i].nom, t[i].valeur);
  printf("</UL>\n");
}
```

4.3.3 Traitement des erreurs

Un programme robuste doit prévoir toutes les erreurs possibles, qu'elles proviennent d'une mauvaise utilisation du programme dans une page HTML, ou de données incorrectes fournies par l'utilisateur. En cas d'erreur, le programme doit produire un document contenant un message d'erreur. Par exemple, après avoir testé que la méthode employée est `POST`, on affichera la réponse suivante si ce n'est pas le cas :

```
  if(strcmp(getenv("REQUEST_METHOD"),"POST")) {
    printf("Content-type: text/html\n\n");
    printf("Ce programme doit être appelé par la méthode POST.\n");
    exit(1);
  }
```

4.3.4 Construction de documents à partir de modèles

Le document construit est généralement *personnalisé* (à la manière du publipostage de Word) à partir d'un document modèle et de données entrées par l'utilisateur ou calculées autrement (par exemple la date, ou un nombre d'accès).

Par exemple, on veut produire un document

```
<HTML>
<BODY>
<H1>Salut, Lolo</H1>
</BODY>
</HTML>
```

où `Salut` et `Lolo` ont été saisis par un formulaire. Il est commode de créer le fichier modèle

```
<HTML>
<BODY>
<H1>$0, $1</H1>
</BODY>
</HTML>
```

où les « variables » `$0`, `$1`, etc, seront remplacés par les chaînes voulues. La fonction `subst_copie` copie sur la sortie standard un fichier donné, après avoir effectué les substitutions induquées dans son premier argument. Elle utilise les fonctions `getc` et `putc` qui, respectivement, lit ou écrit un caractère dans un fichier.

```
#include <stdio.h>

void subst_copie(int p, char *subst[], char *fichier_modele)
{
  FILE *fp = fopen(fichier_modele, "r");
  int c;
  int n, l;

  if (fp == NULL) {
    exit(1);
  }

  c = getc(fp);
  while (c != EOF) {
    if (c != '$') {
      putc(c, stdout);              /* copier un caract. ordinaire */
    } else {
      l = fscanf(fp, "%d", &n);
      if (l==1 && n<p)              /* si un entier a ete lu */
        printf("%s", subst[n]);     /* appliquer la substitution */
      else {
```

```
        putc(c, stdout);            /* copier ce $ */
      }
    }
    c = getc(fp);
  }

  fclose(fp);
}

int main()
{
  couple couples[MAX];
  int m;
  char *longue_chaine = extraire_couples(couples, &m);
  char *subst[2];

  decoder_couples(m, couples);
  subst[0] = couples[0].valeur;
  subst[1] = couples[1].valeur;
  subst_copie(2, subst, "message.mod");
  return 0;
}
```

4.3.5 Lire et écrire dans un fichier : l'exemple d'un compteur

On conserve le nombre d'accès à un document dans un fichier `compteur.txt`. À chaque accès, ce fichier est ouvert, l'entier qu'il contient est lu, incrémenté, puis écrit à nouveau dans le fichier.

Comme il y a lecture et écriture, le fichier doit être ouvert en mode `r+` (deuxième argument de `fopen`) et on passe du mode lecture au mode écriture en le « rembobinant » avec la fonction `rewind`. On n'oublie pas de le refermer avant de sortir.

```
#include <stdio.h>

int main() {
  FILE *fp = fopen("compteur.txt", "r+");
  int nb_acces=1, lu;

  if (fp == NULL) {
    exit(1);
  }
  lu = fscanf(fp, "%d", &nb_acces);
  if (lu == 1) nb_acces++;
  rewind(fp);
  fprintf(fp, "%d", nb_acces);
  fclose(fp);
  return 0;
```

```
}
```

4.4 La compilation du programme CGI

Soit un programme CGI de source `prog.cgi.c`; après compilation, il s'appelle `prog.cgi`.

Ce programme sera exécuté, non par son propriétaire (qui est celui qui l'a compilé) mais par le serveur HTTP, qui n'a aucun droit pour créer ou modifier des fichiers. Si `prog.cgi` doit créer ou modifier des fichiers, il faut lui donner les droits de son propriétaire, à l'aide de la commande :

```
unix% chmod u+s prog.cgi
```

Enfin, on rassemblera les programmes exécutables dans un répertoire particulier auquel le serveur HTTP a accès :

```
unix% mv prog.cgi $(HOME)/www/cgi-bin
```

Il est commode de réunir ces commandes dans une règle du `Makefile` du répertoire où se fait la compilation :

```
prog: prog.cgi.o
        $(CC) -o prog.cgi prog.cgi.o
        chmod u+s prog.cgi
        mv prog.cgi $(HOME)/www/cgi-bin
```

Ces commandes seront exécutées (sous Emacs ou dans un shell) par :

```
make prog
```

Chapitre 5

JavaScript

Le manuel de référence complet et le guide de programmation sont disponibles aux URL
http://cermics.enpc.fr/doc/javascript/ref/
http://cermics.enpc.fr/doc/javascript/guide/

5.1 Qu'est-ce que JavaScript ?

Alors que les programmes CGI permettent de créer des documents dynamiques côté serveur, JavaScript permet de créer des documents dynamiques côté client. Il est nécessaire d'utiliser CGI quand la construction du document utilise des données qui sont nécessairement conservées par le serveur : par exemple un compteur. Par contre, si les données utilisées sont connues du client, par exemple la date courante, on utilise plutôt JavaScript, qui est beaucoup plus récent et plus simple.

JavaScript est :

- un langage de programmation : il a des variables, des fonctions, des opérateurs arithmétiques et booléens, des conditionnelles, des boucles, etc
- interprété par le navigateur : les programmes JavaScript sont intégrés dans les fichiers HTML, ils ne sont pas compilés et leur exécution est très lente comparée à celle d'un exécutable C
- couplé avec HTML : les objets associés au document, ainsi que les événements, sont accessibles par JavaScript

Il existe d'autres langages de scripts : Visual Basic et Tcl. Le type MIME d'un script est « `application/x-javascript` ».

Les programmes JavaScript sont appelés des scripts. Leur exécution peut se produire à deux moments :

- au chargement du document qui contient le script
- sur événement

5.2 Deux exemples

5.2.1 Afficher la date

Le document suivant contient un script

```
<BODY>
 ...
<SCRIPT LANGUAGE="JavaScript">
var jour = new Date()
document.write("Champs sur Marne, le ", jour.getDate(), "/",
               jour.getMonth() + 1, "/", jour.getYear(), "<P>")
</SCRIPT>
...
</BODY>
```

qui, exécuté au chargement du document, a pour effet d'afficher le résultat de la concaténation de plusieurs chaînes : « Champs sur Marne le », jour, mois, année (date connue du navigateur), et d'un `<P>`.

Contrairement à HTML, la casse des lettres importe : il ne faut pas écrire `date` à la place de `Date`, ni `Write` à la place de `write`.

Si le navigateur qui charge ce document ignore JavaScript (Netscape était le premier navigateur à interpréter JavaScript), il va afficher tout ce qui figure entre `<SCRIPT ...>` et `</SCRIPT>` : « `var jour`... ». Pour éviter ce problème, il faut placer le contenu du script dans des commentaires HTML ; de plus, la fin de ce commentaire doit être placée dans un commentaire JavaScript:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
var jour = new Date()
document.write("Champs sur Marne, le ", jour.getDate(), "/",
               jour.getMonth() + 1, "/",jour.getYear(), "<P>")
// -->
</SCRIPT>
```

5.2.2 Alerte sur événement

Quand le navigateur quitte un document dont le `<BODY>` spécifie le traitement d'événement suivant

```
<BODY onUnload="alert('Au revoir', '')">
```

une fenêtre d'alerte est ouverte, contenant le message « Au revoir ».

5.3 Les objets

5.3.1 Objets, propriétés, méthodes

JavaScript est un langage à objets, qui sont voisins des structures de C. Ces objets ont donc des champs, accessibles comme en C, par une notation pointée. Ces champs sont de deux catégories :

- des *propriétés*, qui sont, soit des valeurs (numériques, booléennes, ou chaînes de caractères), soit d'autres objets
- ou des méthodes, qui sont des fonctions qui s'appliquent à l'objet

Par exemple, `"JavaScript"` est un objet dont une propriété est sa longueur, `"JavaScript".length`, qui vaut 10 ; on peut lui appliquer la méthode `substring()` par `"JavaScript".substring(0,3)`, qui retourne la sous-chaîne des quatre premiers caractères, `"Java"`.

5.3.2 Les objets du navigateur

Quand le navigateur charge un document, un certain nombre d'objets liés à ce document lui deviennent accessibles, par exemple :

- `window` : la fenêtre dans laquelle le document est chargé
- `window.location` : l'URL du document chargé dans cette fenêtre
- `window.document` : le contenu dans ce document
- `window.history` : l'historique des URL affichés dans cette fenêtre (les documents passent, la fenêtre reste)

Comme `window` est l'objet principal, il peut être omis.

Prenons l'exemple du document suivant :

```
<BODY>
<FORM NAME="formulaire"
      METHOD="Post"
      ACTION="http://www.enpc.fr/cgi-bin/truc.cgi"
>
<INPUT TYPE="text" NAME="nom"><P>
<INPUT TYPE="button" NAME="bouton1" VALUE="pomme">
<INPUT TYPE="submit" VALUE="Envoyer">
</FORM>
</BODY>
```

Les objets suivants sont connus :

- `document.URL` : l'URL de ce document
- `document.bgColor` : sa couleur de fond
- `document.title` : son titre
- `document.lastModifed` : sa date de dernière modfication
- `document.formulaire` : ce formulaire
- `document.formulaire.action` : l'action exécutée
- `document.formulaire.length` : le nombre de composants, 3
- `document.formulaire.bouton1` : le bouton
- `document.formulaire.bouton1.value` : la chaîne affichée sur ce bouton

– `document.formulaire.nom` : le champ textuel
– `document.formulaire.nom.value` : la chaîne écrite par l'utilisateur dans ce champ

5.3.3 Événements

JavaScript permet de traiter les événements qui se produisent dans le navigateur : par exemple, le chargement d'une page, le positionnement du pointeur sur un lien, la soumission d'un formulaire, le pressage d'un bouton, etc.

L'action à réaliser quand un événement est produit doit être spécifiée dans un nouveau type de champ des marquages HTML, commençant par `on...`, par exemple `onLoad`, `onMouseOver`, `onSubmit`, `onClick`, etc. Une instruction quelconque peut être affectée à ce champ d'événement.

Un événement est un nouveau type d'objet, avec ses propriétés : `target`, `type`, `data`, `height`, `layerX`, `layerY`, `modifiers`, `pageX`, `pageY`, `screenX`, `screenY`, `which`, `width`.

Pour que le traitement d'un événement puisse accéder aux propriétés de cet événement, il faut affecter à l'attribut de traitement une fonction d'une variable qui sera appliquée à l'événement courant. L'exemple suivant affiche dans une fenêtre d'alerte le type (qui est `mousedown`) et la position dans la page de l'événement déclenché quand un bouton de souris est pressé.

```
<SCRIPT>
function souris(evnt) {
   alert ("Le document a reçu un événement : " + evnt.type +
          " en x = " + evnt.layerX +
          " et y = " + evnt.layerY
         );
   return true;
   }
document.onMouseDown = souris;
</SCRIPT>
```

5.3.4 Cadres

JavaScript est particulièrement utile pour manipuler les différents panneaux d'un cadre. Voici un document qui définit un cadre à deux panneaux, chacun étant nommé :

```
<FRAMESET ROWS="20%,90%">
  <FRAME NAME="tdm" SRC="tdm.html">
  <FRAME NAME="texte" SRC="introduction.html">
</FRAMESET>
```

À la fenêtre initiale qui charge ce document, va succéder deux fenêtres, `tdm` et `texte`, chacune chargeant un document ; ces deux fenêtres ont comme *parente* la fenêtre initiale.

Dans le fichier `tdm.html`, l'objet désigné par `parent` est cette fenêtre parente ; `parent.tdm` et `parent.texte` sont les deux panneaux. Ainsi, on pourra accéder, dans le fichier `tdm.html` à un objet du deuxième panneau, par exemple `parent.texte.document.title`.

5.3.5 La hiérarchie des fenêtres

```
<FRAMESET COLS="50,*">
  <FRAME NAME="tdm" SRC="tdm.html" NORESIZE SCROLLING="no">
  <FRAMESET ROWS="20%,70%,10%">
    <FRAME NAME="titre" SRC="titre.html" NORESIZE>
    <FRAME NAME="texte" SRC="chap1.html">
    <FRAME NAME="navig" SRC="navig.html" NORESIZE>
  </FRAMESET>
</FRAMESET>
```

Ces quatre panneaux ont la même fenêtre parente (indépendamment de l'imbrication des `<FRAMESET ... >`.

5.3.6 Ouvrir simultanément plusieurs documents

Quand on travaille avec plusieurs fenêtres (notamment des panneaux), on peut vouloir ouvrir simultanément plusieurs documents en suivant un seul lien. Les scripts suivants réalisent ceci à l'aide d'un bouton :

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function suivre_2_liens(page1, page2) {
  parent.tdm.location = page1;
  parent.texte.location = page2;
}
</SCRIPT>
</HEAD>

<BODY>
<FORM NAME="boutons">
<INPUT TYPE = "button"
       VALUE = "Chapitre 1"
       onClick = "suivre_2_liens('tdm1.html','chap1.html')"
>
</FORM>
</BODY>
</HTML>
```

Avec une ancre à la place d'un bouton :

```
<A HREF="tdm1.html"  TARGET = "tdm"
   onClick = "parent.texte.location = 'chap1.html'">
Chapitre 1</A>
```

On peut aussi utiliser une URL dont le *protocole* est `javascript` (au lieu de `http`, `mailto` ou `ftp`, etc), contenant une expression JavaScript.

```
<A HREF="javascript:suivre_2_liens('tdm1.html','chap1.html')">Chapitre 1</A>
```

5.3.7 Écrire sur la ligne de statut de la fenêtre

On peut remplacer l'URL d'un lien qui s'affiche dans la ligne du bas du navigateur quand le curseur se trouve sur l'ancre, par un autre message, de la façon suivante :

```
<A HREF="mailto:rl@enpc.fr"
   onMouseOver="window.status='Ecrivez-moi'; return true">
```

5.4 Traitement des formulaires

5.4.1 Radio

Un groupe de boutons à choix exclusif doit porter le même nom.

```
<FORM NAME="f" ...>
<INPUT TYPE="radio" NAME="q4" VALUE="0" onClick="..."> Non <BR>
<INPUT TYPE="radio" NAME="q4" VALUE="1" onClick="..."> Oui <BR>
</FORM>
```

Ces deux boutons radio sont désignés en JavaScript par `document.f.q4[0]` et `document.f.q4[1]`. La propriété `checked` indique si le bouton est sélectionné : `document.f.q4[0].checked` vaut `true` ou `false`. L'événement `onClick` permet d'exécuter une action quand ce bouton est sélectionné.

5.4.2 Checkbox

Contrairement aux boutons radio, chaque checkbox a son nom.

5.4.3 Select

Chaque liste de sélection, dans un formulaire, est un objet. Ses propriétés sont :

- `name` : son mom
- `options` : le tableau de ses options, désignées par `options[0]`, `options[1]`, etc.
- `length` : le nombre de ses options
- `selectedIndex` : l'indice de l'option sélectionnée (ou de la première sélectionnée en cas de sélection multiple).

Chaque option d'une liste de sélection est un objet. Ses propriétés sont :

- `name` : son nom, qui est la valeur de l'attribut `NAME`

- `index` : son indice
- `text` : le texte affiché
- `value` : la valeur de l'attribut `VALUE`
- `selected` : booléen qui indique si l'option est sélectionnée

L'événement `onChange` permet de traiter la sélection faite par l'utilisateur.

5.4.4 Exemple : additionner deux champs

```
<FORM>
A = <INPUT TYPE = "text" NAME = "A"><BR>
B = <INPUT TYPE = "text" NAME = "B"><BR>
<INPUT TYPE = "button" VALUE = "Additionner"
       onClick = "this.form.S.value =
                    parseInt(this.form.A.value) +
                    parseInt(this.form.B.value)"
<INPUT TYPE="text" NAME="S">
```

5.4.5 Définir une fonction de traitement

On définira les fonctions dans l'en-tête de la page HTML ; ce qui figure dans l'en-tête est évalué avant que le contenu du document ne soit affiché.

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function calculer(forme)
{
  forme.S.value = parseInt(forme.A.value) + parseInt(forme.B.value)
}
</SCRIPT>
</HEAD>
```

Dans le champ d'événement du bouton, on écrira simplement :

```
<INPUT TYPE = "button" VALUE = "Additionner"
       onClick = "calculer(this.form)">
```

5.5 Animation graphique

5.5.1 Temporisations

La méthode `setTimeout` d'une fenêtre permet d'évaluer une expression à l'expiration d'un délai (spécifié en millisecondes) :

```
afaire = setTimeout(tropTard(), 10000)
```

On peut l'utiliser par exemple pour laisser un certain temps à l'utilisateur pour répondre à un formulaire ; une fois le temps écoulé, la fonction `tropTard()`, définie par l'utilisateur (par exemple un appel à `alert()`), sera exécutée.

La méthode `clearTimeout` annule le compte à rebours. Une fonction de traitement local du questionnaire (par exemple traitant l'événement `onSubmit`) devra exécuter

```
clearTimeout(afaire)
```

pour empêcher l'exécution de `tropTard()`.

5.5.2 Images animées

Les images contenues dans un document sont accessibles comme des propriétés du document. En modifiant l'URL d'une image, on peut facilement faire défiler une suite d'images.

```
<SCRIPT language="JavaScript">
var temps = 100, numero = 1

// Chargement des images
lesImages = new Array()
for (i = 1; i < 6; i++) {
   lesImages[i] = new Image()
   lesImages[i].src = "images/animation" + i + ".gif"
}

function animer() {
   document.anim.src = lesImages[numero].src
   numero = (numero < 5) ? numero + 1 : 1
}
</SCRIPT>

<IMG NAME="anim" SRC="images/critic1.gif" ALT="[Animation]"
        onLoad="setTimeout('animer()', temps)">
```

5.6 Fenêtres et documents dynamiques

Une fenêtre, la *fenêtre principale*, est ouverte quand le navigateur est lancé ; quand un document, qui consiste en la définition d'un cadre (`<FRAMESET>` est chargé, une fenêtre est créée pour chacun des panneaux (`<FRAME>`) spécifié. Quand l'attribut `TARGET` d'un lien a pour valeur `_blank`, une fenêtre indépendante est également créée.

Toutes les façons précédentes de créer une fenêtre sont du ressort d'HTML. Il y a d'autres moyens de créer des fenêtres grâce à JavaScript.

5.6.1 Fenêtres de dialogue

Les méthodes `alert`, `confirm` et `prompt`, de l'objet `window` créent des fenêtres indépendantes spéciales :

```
alert("Erreur : cette réponse n'est pas permise")
confirm("voulez-vous vraiment quitter cette page ?")
prompt("Quel est votre nom ?", nom)
```

où `nom` est une variable ou une propriété JavaScript.

5.6.2 Création de fenêtres

Les méthodes `open` et `close` permettent de créer et de fermer une fenêtre :

```
var fenetre = open("truc.html", "truc")
```

La valeur retournée par la méthode `open` est affectée à la variable `fenetre`, qui aura les propriétés et les méthodes des objets de type `window`.

Le premier argument est une URL qui désigne le document qui doit être chargé dans cette fenêtre ; si la chaîne est vide, le contenu du document devra être créé. Le deuxième argument donne à la fenêtre un nom qui pourra être utilisé comme valeur de l'attribut `TARGET`. Un troisième argument, optionnel, est une chaîne de caractères qui spécifie des propriétés de la fenêtre créée, par exemple :

```
var fenetre = open("",
                   "truc",
                   "width=300,height=100,status,scrollbar")
```

5.6.3 Création d'un document

Un script contenu dans un document ne peut agir sur ce document qu'en altérant la valeur des propriétés de ce document. On peut ainsi modifier la couleur de fond, les formulaires ou les images, mais pas le texte, car ce n'est pas une propriété.

Addition de texte

La seule façon d'opérer sur le texte d'un document est d'y ajouter du texte, ce qui sera exécuté une seule fois, au chargement du document :

```
<SCRIPT LANGUAGE="JavaScript">
document.write("<FONT SIZE=-1>Derni&egrave;re mise à jour : " +
               document.lastModified + "</FONT>")
</SCRIPT>
```

Modifications interactives des propriétés

L'aspect interactif est résolu par la capacité de JavaScript de traiter des événements.

Documents dynamiques

Il s'agit de l'équivalent, côté client, des documents construits par un programme CGI, côté serveur.

Les méthodes `open`, `write` et `close`, appliquées à un document, permettent de créer un document, d'y écrire et de le fermer. Le contenu du document peut ainsi être calculé (à partir de données saisies dans un formulaire, la date courante, etc).

```
var fenetre = open("", "truc")
fenetre.document.open("text/html")
fenetre.document.write("<HTML> ... </HTML>")
fenetre.document.close()
```

5.7 Couches dynamiques

Si un document définit des couches à l'aide de styles nommés, les propriétés de ces couches sont accessibles par JavaScript. Ainsi, avec le style

```
<STYLE>
#couche1 {
  position: absolute;
  top: 2cm;
  left: 2cm;
  width: 6cm;
  z-index: 1;
}
#couche2 {
  position: absolute;
  top: 3cm;
  left: 3cm;
  width: 6cm;
  z-index: 1
}
</STYLE>
```

Les propriétés `document.couche1` et `document.couche2` sont créées. Ce sont des objets qui ont les propriétés suivantes :

- `document` : le contenu de la couche (qui figurera par exemple dans un `<SPAN ID="couche1"> ... </SPAN>`)
- `left`, `top` : position relative à la couche parente
- `pageX`, `pageY` : position absolue (relative à la fenêtre)

- `zIndex`
- `bgColor`
- `src` : l'URL du document chargé dans la couche

Les méthodes suivantes s'appliquent :

- `moveBy(dx, dy)`
- `moveTo(x,y)`
- `moveToAbsolute(x,y)`
- `resizeTo(width, height)`
- `load(sourcestring, width)`

Un objet couche peut être créé dynamiquement par

```
couche3 = new Layer(200)            // 200 est la largeur
couche4 = new Layer(200, couche3)   // couche3 est parente de couche4
```

Après l'avoir créée, les diverses propriétés de la couche peuvent être spécifiées.

Une couche a les mêmes capacités de traitement d'événement qu'un document séparé. Par exemple, la couleur de fond d'une couche peut être modifiée par le *script local* suivant :

```
<SPAN ID="couche1">
<P>Une couche de couleur variable</P>

<SCRIPT>
 function changeColor(newcol) {
   bgColor=newcol;
   return false;
 }
 function onMouseOver() {changeColor("blue");}
 function onMouseOut() {changeColor("red");}
</SCRIPT>
</SPAN>
```